WEST

Help | Logout | Interrupt

Main Menu | Search Form | Posting Counts | Show S Numbers | Edit S Numbers | Preferences | Cases

Search Results -

Terms	Documents
L4 and l1	34

Database:
US Patents Full-Text Database
US Pre-Grant Publication Full-Text Database
JPO Abstracts Database
EPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search: L5

Refine Search

Recall Text | Clear

Search History

DATE: Wednesday, August 27, 2003 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
	DB=USPT,PGPB,JPAB,EPAB,DWPI,TDBD; PLUR=YES; OP=ADJ		
L5	L4 and l1	34	L5
L4	L3 same map$4	143	L4
L3	graphics adj3 (device or engine or circuit$ or module or logic or unit or section or block or $5processor or accelerator) near5 memory same (texture or textel or coordinate or tile)	418	L3
L2	L1	873	L2
L1	(frame buffer or memory) near5 (texture data or coordinate or tile) same (vertex or vertices or primitive or polygon or triangle)	873	L1

END OF SEARCH HISTORY

WEST

Generate Collection | Print

Search Results - Record(s) 1 through 25 of 25 returned.

1. Document ID: US 20030095338 A1

L9: Entry 1 of 25 File: PGPB May 22, 2003

DOCUMENT-IDENTIFIER: US 20030095338 A1
TITLE: System and method for panoramic imaging

Detail Description Paragraph (80):
[0126] FIG. 21 is a flow diagram that illustrates a particular example of the processing method. At the start of the process, as illustrated in block 196, a warped source image is chosen as shown in block 198 from a warped image source 200. Several processes are performed to unwarp the image. In particular, block 202 shows that the warped image is "captured" by a video frame grabber, and block 204 shows that the pixel data from the source image is transferred to a texture map memory buffer as a texture map. Block 206 shows that a user or pre-determined meta-data can identify a particular virtual model to use, and block 208 shows that a user or pre-determined meta-data can identify a particular projection to use. In block 210 the vertices are produced for the virtual model, and in block 212 the projection is set up by computing the texture map coordinates for the vertices of the virtual model. Next, the virtual model is transferred to a graphics hardware device by transferring the vertex coordinates as shown in block 214 and transferring the texture map coordinates as shown in block 216. Block 218 shows that video is now ready to be displayed. In particular, block 220 shows that the renderer may spawn multiple and simultaneous threads to display the video. At block 222, the render can determine if the user has entered particular viewing parameters, such as zooming or the particular portion of the panorama to view, as shown in block 224, and instruct the hardware to make the appropriate corrections to the virtual model. Back at block 204 the renderer can make the pixel data of the current texture map from the texture map memory buffer available to the graphics hardware device, and at block 202 the renderer can instruct the software to "capture" the next video frame and map that pixel data to the texture map memory buffer as a new texture map at block 204. The graphics hardware device will use the pixel data from the texture map memory buffer to complete the virtual model, and will update the display by displaying the completed virtual model as a viewable panoramic image as shown at block 226. In one embodiment, the graphics hardware device may utilize an interpolation scheme to, "fill" in the pixels between the vertices and complete the virtual model. In this embodiment, a barycentric interpolation scheme could be used to calculate the intermediate values of the texture coordinates between the vertices. Then, a bilinear interpolation scheme could be used on the source pixels residing in the texture map to actually transfer the appropriate source pixel into the appropriate location on the model. The renderer can continue these procedures in a continuous loop until the user instructs the process to stop, or there is no longer any pixel data from the warped image source. FIG. 21 also shows that direct memory access (DMA) can be utilized if the hardware will support it. DMA can be used, for example, in allowing the texture map from the captured video frame to be directly available for the graphics hardware device to use.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | Claims | KWIC | Draw Desc | Image

2. Document ID: US 20030095131 A1

L9: Entry 2 of 25 File: PGPB May 22, 2003

DOCUMENT-IDENTIFIER: US 20030095131 A1

TITLE: Method and apparatus for processing photographic images

Detail Description Paragraph (24):
[0049] FIG. 3 is a flow diagram that illustrates a particular example of the processing method. At the start of the process, as illustrated in block 32, a warped source image is chosen as shown in block 34 from a warped image source 36. Several processes are performed to unwarp the image. In particular, block 38 shows that the warped image is "captured" by a video frame grabber, and block 40 shows that the pixel data from the source image is transferred to a texture map memory buffer as a texture map. Block 42 shows that a user or predetermined meta-data can identify a particular virtual model to use, and block 44 shows that a user or pre-determined meta-data can identify a particular projection to use. In block 46 the vertices are produced for the virtual model, and in block 48 the projection is set up by computing the texture map coordinates for the vertices of the virtual model. Next, the virtual model is transferred to a graphics hardware device by transferring the vertex coordinates as shown in block 50 and transferring the texture map coordinates as shown in block 52. Block 54 shows that video is now ready to be displayed. In particular, block 56 shows that the renderer may spawn multiple and simultaneous threads to display the video. At block 58, the renderer can determine if the user has entered particular viewing parameters, such as zooming or the particular portion of the panorama to view, as shown in block 60, and instruct the hardware to make the appropriate corrections to the virtual model. Back at block 40 the renderer can make the pixel data of the current texture map from the texture map memory buffer available to the graphics hardware device, and at block 38 the renderer can instruct the software to "capture" the next video frame and map that pixel data to the texture map memory buffer as a new texture map at block 40. The graphics hardware device will use the pixel data from the texture map memory buffer to complete the virtual model, and will update the display by displaying the completed virtual model as a viewable panoramic image as shown at block 62. In one embodiment, the graphics hardware device may utilize an interpolation scheme to "fill" in the pixels between the vertices and complete the virtual model. In this embodiment, a barycentric interpolation scheme could be used to calculate the intermediate values of the texture coordinates between the vertices. Then, a bilinear interpolation scheme could be used on the source pixels residing in the texture map to actually transfer the appropriate source pixel into the appropriate location on the model. The renderer can continue these procedures in a continuous loop until the user instructs the process to stop, or there is no longer any pixel data from the warped image source. FIG. 3 also shows that direct memory access (DMA) can be utilized if the hardware will support it. DMA can be used, for example, in allowing the texture map from the captured video frame to be directly available for the graphics hardware device to use. As noted above, the renderer may execute some of the steps simultaneously. Therefore, it is to be understood that the steps shown in the flow diagram of FIG. 3 may be not necessarily be performed in the exact order as shown and described.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | Claims | KWIC | Draw Desc | Image

---

☐ 3. Document ID: US 20030068098 A1

L9: Entry 3 of 25 File: PGPB Apr 10, 2003

DOCUMENT-IDENTIFIER: US 20030068098 A1
TITLE: System and method for panoramic imaging

Detail Description Paragraph (89):
[0137] FIG. 27 is a flow diagram that illustrates a particular example of the processing method. At the start of the process, as illustrated in block 244, a warped source image is chosen as shown in block 246 from a warped image source 248. Several processes are performed to unwarp the image. In particular, block 250 shows that the warped image is "captured" by a video frame grabber, and block 252 shows that the pixel data from the source image is transferred to a texture map memory buffer as a texture map. Block 254 shows that a user or pre-determined meta-data can identify a particular virtual model to use, and block 256 shows that a user or pre-determined meta-data can identify a particular projection to use. In block 258 the vertices are produced for the virtual model, and in block 260 the projection is set up by computing the texture map coordinates for the vertices of the virtual model. Next, the virtual model is transferred to a graphics hardware device by transferring the vertex coordinates as shown in block 262 and transferring the texture map coordinates as shown in block 264. Block 266 shows that video

is now ready to be displayed. In particular, block 268 shows that the renderer may spawn multiple and simultaneous threads to display the video. At block 270, the render can determine if the user has entered particular viewing parameters, such as zooming or the particular portion of the panorama to view, as shown in block 272, and instruct the hardware to make the appropriate corrections to the virtual model. Back at block 252 the renderer can make the pixel data of the current texture map from the texture map memory buffer available to the graphics hardware device, and at block 250 the renderer can instruct the software to "capture" the next video frame and map that pixel data to the texture map memory buffer as a new texture map at block 252. The graphics hardware device will use the pixel data from the texture map memory buffer to complete the virtual model, and will update the display by displaying the completed virtual model as a viewable panoramic image as shown at block 274. In one embodiment, the graphics hardware device may utilize an interpolation scheme to "fill" in the pixels between the vertices and complete the virtual model. In this embodiment, a barycentric interpolation scheme could be used to calculate the intermediate values of the texture coordinates between the vertices. Then, a bilinear interpolation scheme could be used on the source pixels residing in the texture map to actually transfer the appropriate source pixel into the appropriate location on the model. The renderer can continue these procedures in a continuous loop until the user instructs the process to stop, or there is no longer any pixel data from the warped image source. FIG. 27 also shows that direct memory access (DMA) can be utilized if the hardware will support it. DMA can be used, for example, in allowing the texture map from the captured video frame to be directly available for the graphics hardware device to use.

☐ 4. Document ID: US 20030001857 A1

L9: Entry 4 of 25 File: PGPB Jan 2, 2003

DOCUMENT-IDENTIFIER: US 20030001857 A1
TITLE: Method and apparatus for determining logical texture coordinate bindings

Abstract Paragraph (1):
A computer system and method are provided for mapping of texture images. This may include a memory device to store a plurality of texture coordinates associated with vertices of three dimensional objects and a graphics device coupled to the memory device to process internal texture coordinates. A mapping system may appropriately route select ones of the plurality of texture coordinates from the memory device to the graphics device. The texture images may be mapped onto objects that, when rendered, include the image that may later be displayed on a display device.

Detail Description Paragraph (16):
[0028] Embodiments of the present invention will be described with respect to a computer system that includes a memory device to store a plurality of texture coordinates associated with vertices of three dimensional objects, a graphics device to couple to the memory device and to process internal texture coordinates for display, and a mapping system to appropriately route select ones of the plurality of texture coordinates from the memory device to the graphics device.

Detail Description Paragraph (18):
[0030] During texture mapping, select information of the array 250 may be passed from the memory (i.e., the system memory 130) to a graphics device (i.e., the GMCH 140) to appropriately prepare the image. However, it may be disadvantageous to transfer the whole array 250 from the memory to the graphics device as the array 250 may contain an extremely large amount of data and may need to be reformatted at the graphics device. Additionally, hardware (i.e., the graphics device) may only be capable of storing a predetermined number of texture coordinates at one time. For example, the graphics device may only be capable of simultaneously storing data regarding four separate texture coordinates. Disadvantageous arrangements may require the use of software in the transfer of the array 250 to the graphics device. Some of the texture coordinates may be stripped from the array 250 after the array 250 has been transferred to the graphics device. This may involve the software copying the array 250 to an intermediate buffer and then passing select information in the intermediate buffer to the mapping engines. It is therefore desirable for a method and apparatus to transfer only select portions of the array 250 that will be used in

the texture engines of the graphics device.

Detail Description Paragraph (21):
[0033] FIG. 5 illustrates texture coordinate data transfer according to an example embodiment of the present invention. Other embodiments and configurations are also within the scope of the present invention. More specifically, FIG. 5 illustrates a memory 260 (such as the system memory 130 of FIG. 3), a graphics device 270 (such as the GMCH 140 of FIG. 3) and software 280 that controls, among other things, the transfer of texture coordinates from the memory 260 to the graphics driver 270. The software 280 may reside in external memory. The memory 260 is shown as including the array 250. For illustration purposes, the array 250 may include texture coordinate data 252 and non-texture coordinate data 254. As show in FIG. 5, the graphics device 270 may include four texture mapping engines, namely a texture mapping engine (TME0) 272, a texture mapping engine (TME1) 274, a texture mapping engine (TME2) 276, and a texture mapping engine (TME3) 278. While this embodiment only illustrates four texture mapping engines within the graphics device 270, other numbers of texture mapping engines are also within the scope of the present invention.

Detail Description Paragraph (22):
[0034] The graphics device 270 may further include a register 271 associated with the texture mapping engine 272, a register 271 associated with the texture mapping engine 274, a register 275 associated with the texture mapping engine 276 and a register 277 associated with the texture mapping engine 278. Each of the registers 271, 273, 275 and 277 may be used to select the appropriate texture coordinate values to be obtained from the memory 260 (or a default value) for each of the texture mapping engine 272, the texture mapping engine 274, the texture mapping engine 276 and the texture mapping engine 278, respectively. During operation, the software 280 may set values within the respective registers 271, 273, 275 and 277 such that the texture mapping engines 272, 274, 276 and 278 receive the appropriate texture coordinates from the memory 260. In accordance with embodiments of the present invention, the entire array 250 of texture coordinates does not needed to be transferred from the memory 260 to the graphics device 270. That is, embodiments of the present invention route the proper texture coordinates to the appropriate texture mapping engines and avoid transferring unneeded texture coordinates from the array 250. The software 280 may appropriately pick the texture coordinates to be transferred to the texture mapping engines 272, 274, 276 and 278. This avoids the graphics device 270 from having to reformat the vertex texture buffers after they have been transferred from the memory 260 to the graphics device 270.

Detail Description Paragraph (23):
[0035] FIGS. 6 and 7 illustrate how embodiments of the present invention may be useful to appropriately route the texture coordinate data from the memory 260 to the graphics device 270. As shown in FIG. 6, the memory 260 (of FIG. 5) may include eight vertex texture coordinates TC0, TC1, TC2, TC3, TC4, TC5, TC6 and TC7. These texture coordinates may be provided within the array 250 or may be provided within vertex texture buffers. The graphics device 270 (of FIG. 5) is represented in FIG. 6 as four internal texture coordinates (ITC0, ITC1, ITC2 and ITC3) that will be provided within the four texture mapping engines 272, 274, 276 and 278, respectively. In accordance with embodiments of the present invention, the four internal texture coordinates (ITC0, ITC1, ITC2 and ITC3) to be provided to the four texture mapping engines 272, 274, 276 and 278 (at the graphics device 270) may be default values 290 (such as 0,0,0) or may be one of the texture coordinates TC0, TC1, TC2, TC3, TC4, TC5, TC6 and TC7 provided within the memory 260. The selection as to which texture coordinates will be provided as the internal texture coordinates ITC0, ITC1, ITC2 and ITC3 (corresponding to the mapping engines 272, 274, 276 and 278) may be based on a signal TexCoordbinding 295. This signal may be provided by the software 280 to appropriately route the appropriate texture coordinates to the texture mapping engines 272, 274, 276 and 278. That is, the software 280 may specify, in this example, that the internal texture coordinates may come from any one of the texture coordinates TC0, TC1, TC2, TC3, TC4, TC5, TC6 and TC7 or from the default value. In other words, the software 280 may select the source of the texture coordinates for each texture mapping engine 272, 274, 276 and 278. The software 280 may make this selection based on the desired image to be created and the desire to avoid unneedlessly transferring data from the memory 260 to the graphics device 270. The apparatus appropriately routes (or transfers) the selected texture coordinates to the proper texture mapping engines 272, 274, 276 and 278 (as the internal texture coordinates ITC0, ITC1, ITC2 and ITC3) by utilizing the registers 271, 273, 275 and 277. Accordingly, the graphics device 270 may only obtain a limited number of texture coordinates from the memory 260 and avoid obtaining the unnecessary texture coordinates for a particular image. This additionally avoids the software 280 from having to reformat the array 250 when it arrives at the graphics device 270 as in disadvantageous arrangements.

CLAIMS:

1. A computer system comprising: a memory device to store a plurality of texture coordinates associated with vertices of three dimensional objects; a graphics device to couple to said memory device and to process internal

texture coordinates for display; and a mapping system to appropriately route select ones of said plurality of texture coordinates from said memory device to said graphics device.

8. A computer system comprising: a memory device to store a plurality of texture coordinates associated with vertices of three dimensional objects; a graphics device having a plurality of mapping engines each to map at least one of said objects based on a plurality of internal texture coordinates; and a mapping system to transfer select ones of said plurality of texture coordinates in said memory device to said mapping engines without transferring unselected ones of said plurality of texture coordinates from said memory device to said graphics device.

13. A graphics device for creating an image based on internal texture coordinates received from a memory device, said graphics device including a plurality of mapping engines and a plurality of registers, each register corresponding to a source of texture coordinate values for one of said mapping engines.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

---

5. Document ID: US 20010045955 A1

L9: Entry 5 of 25 File: PGPB Nov 29, 2001

DOCUMENT-IDENTIFIER: US 20010045955 A1
TITLE: IMAGE GENERATING METHOD AND APPARATUS

Detail Description Paragraph (10):
[0065] The graphic processor comprises a texture mapping circuit 31 which uses an input texture information and three vertex information of a triangle for texture mapping. It should be noted that the texture information (Rt, Gt, Bt, At) are supplied from a video memory (VRAM) incorporated in a system in which the graphic processor is used. Also, note that the three vertex information of a triangle are supplied from a CPU incorporated in a system in which the graphic processor is used.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

---

6. Document ID: US 6597363 B1

L9: Entry 6 of 25 File: USPT Jul 22, 2003

DOCUMENT-IDENTIFIER: US 6597363 B1
TITLE: Graphics processor with deferred shading

CLAIMS:

1. A graphics rendering system for forming a rendered image from graphics primitives, the rendered image stored in a frame buffer, the graphics rendering device comprising: (A) a host processor; (B) a system memory; (C) a frame buffer storing pixels; (D) a device for receiving commands comprising: (1) a 2D command queue memory receiving and storing first direct memory access commands and first commands to perform two-dimensional graphics operations; (2) a 3D command queue memory receiving and storing second direct memory access commands and second commands to perform three-dimensional graphics operations, the second commands comprising: some of the graphics primitives; and part of a pipeline state; (3) a 2D response queue memory receiving and storing third

commands to perform two-dimensional graphics operations; (4) a 3D response queue memory receiving and storing fourth commands to perform three-dimensional graphics operations, the fourth commands comprising: some of the graphics primitives; and part of the pipeline state; and (5) a direct memory access controller coupled to the 2D command queue memory and the 3D command queue memory, the direct memory access controller comprising: (a) logic receiving the stored first direct memory access commands and performing the first direct memory access commands by reading the third commands from the system memory and writing the read third commands into the 2D response queue; and (b) logic receiving the stored second direct memory access commands and performing the second direct memory access commands by reading the fourth commands from the system memory and writing the read fourth commands into the 3D response queue. (E) a geometry unit comprising: (1) logic transforming the graphics primitives; (2) logic clip testing the graphics primitives to determine if the graphics primitives are at least partially contained in a view volume; (3) logic performing face determination to discard any of the graphics primitives that are facing away from a viewing point; (F) a sort memory storing: (1) the part of the pipeline state needed for hidden surface removal; and (2) the part of the graphics primitives needed for hidden surface removal, comprising vertex locations; (G) a state memory storing: (1) the graphics primitives for a frame with attributes of the graphics primitives; and (2) the part of the graphics primitives not needed for hidden surface removal, comprising vertex colors and texture coordinates; (H) a sort unit comprising: (1) logic sorting the graphics primitives according to tile areas within the image; (2) logic reading the sorted graphics primitives from the sort memory according to tile; (I) a setup unit computing derivatives for the sorted graphics primitives; (J) a cull unit performing hidden surface removal to determine which portions of the graphics primitives stored in the scene memory affect the final color of the pixels in the frame buffer; (K) one or more computational units generating fragment color for each sample or group of samples within each of the pixels, the generation of color values for each of the pixels being done only for the portions of the graphics primitives that are determined to affect the pixels, the computational units comprising: (1) a device for interpolating data associated with the graphics primitives, the device comprising: (a) a memory storing cache fill polygon color data, the color data comprising vertex coordinates, w coordinates, and texture coordinates; (b) logic computing barycentric coefficients for each of the graphics primitives from the vertex coordinates and the w coordinates; (c) a cache memory storing a plurality of data cache entries corresponding to a plurality of the graphics primitives, the cached data comprising barycentric coefficients and texture coordinates; (d) a register receiving a fragment of one of the graphics primitives, the fragment comprising fragment coordinates and a sample mask, the sample mask indicating sample locations within the pixel covered by the fragment; (e) a lookup table determining lower bits of an interpolation location, an input of the lookup table comprising the sample mask, the interpolation location having sub-pixel accuracy; (f) logic selecting and reading one of the stored data cache entries corresponding to the received fragment; (g) logic computing interpolation coefficients from the barycentric coefficients and the interpolation location; and (h) logic interpolating the texture coordinates using the computed interpolation coefficients, generating texture coordinates at the interpolation location; (2) a device for interpolating vertex surface normals comprising: (a) logic decomposing each of the vertex surface normals into a vertex magnitude vector and a vertex direction vector; (b) logic computing an interpolated magnitude vector from a plurality of the vertex magnitude vectors and the computed interpolation coefficients; (c) logic computing an interpolated direction vector from a plurality of the vertex direction vectors and the computed interpolation coefficients; and (d) logic combining the interpolated magnitude vector and the interpolated direction vector to generate an interpolated surface normal. (L) a pixel unit comprising: (1) logic blending the fragment colors together to generate per-pixel color values; and (2) logic storing the generated per-pixel color values into the frame buffer; (M) a status monitoring device comprising: (1) logic associating object tags with the three-dimensional objects as the three-dimensional objects are processed by the graphics rendering system; (2) logic testing the three-dimensional objects with one or more visibility criteria as the three-dimensional objects are processed in the graphics rendering system; (3) a status register comprising a plurality of bits; (4) logic selecting one of the bits in the status register according to one of the object tags; (5) logic setting the value of the selected bit in the status register according to response of the corresponding one of the three-dimensional objects to the testing with the visibility criteria; and (6) logic reading and transmitting the status register contents to the host processor, the host processor determining which of the three-dimensional objects are at least partially visible; (N) a device for storing a digital image into the frame buffer comprising: (1) a first memory storing a window identifier map, the window identifier map comprising a window identifier for each of the pixels in the frame buffer; (2) a second memory storing a plurality of bounding box descriptions, each of the bounding box descriptions comprising maximum and minimum pixel coordinates of the corresponding bounding box and a bounding box window identifier; (3) a register storing a programmatically assigned image identifier associated with the digital image; (4) logic selecting an image pixel from the digital image; (5) pixel ownership logic operating on the selected image pixel, the pixel ownership logic comprising: (a) window identifier map test logic enabled by a programmatically set control bit, the window identifier map test logic comprising: (i) logic reading the window identifier from the first memory that corresponds to the selected image pixel; (ii) a comparator circuit comparing the assigned image identifier to the read window identifier; and (iii) logic discarding the selected image pixel if the assigned image identifier does not match the read window identifier; and (b) bounding box test logic enabled by a programmatically set control bit, the

bounding box test logic comprising: (i) logic selecting a first one of the bounding box descriptions such that the selected pixel is within the maximum and minimum pixel coordinates of the selected first bounding box description; (ii) a comparator circuit comparing the assigned image identifier to the bounding box window identifier of the first bounding box description; and (iii) logic discarding the selected image pixel if the assigned image identifier does not match the bounding box window identifier of the first bounding box; and (c) logic writing the selected image pixel to the frame buffer if the selected image pixel was not discarded. (O) a scanout device comprising: (1) logic generating a histogram of the pixel values in the rendered image as the pixel values are stored into the frame buffer; (2) logic computing a transformation function, based on the histogram, to adjust the scale and dynamic range of the pixel values so as to match the digital-to-analog converter; (3) logic reading the pixel values from the frame buffer in a raster line order; (4) logic transforming the read pixel values according to the computed transformation function; and (5) logic inputting the transformed pixel values to the digital-to-analog converter. (P) a scanout device comprising: (1) a memory storing programmatically downloaded interpolation coefficients; (2) a register storing a programmatically selected one of a plurality of fixed zoom ratios; (3) a register storing a programmatically selected location of a zoom bounding box within the computer display; (4) a register storing a programmatically assigned a window identifier associated with the zoom bounding box; (5) logic reading a first pixel color from the frame buffer; (6) logic reading a plurality of second pixel colors from the frame buffer; (7) logic computing a zoomed pixel color from the plurality of second pixel colors and the downloaded interpolation coefficients; (8) logic reading a pixel window identifier from the frame buffer; (9) logic sending the computed zoomed pixel color to the digital-to-analog converter if the display pixel is within the zoom bounding box and the pixel window identifier equals the window identifier assigned to the zoom bounding box; and (10) logic sending the read first pixel color to the digital-to-analog converter if the display pixel is not within the zoom bounding box or the pixel window identifier does not equal the window identifier assigned to the zoom bounding box.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

---

☐ 7. Document ID: US 6556199 B1

L9: Entry 7 of 25 File: USPT Apr 29, 2003

DOCUMENT-IDENTIFIER: US 6556199 B1
TITLE: Method and apparatus for fast voxelization of volumetric models

Detailed Description Text (24):
Preferably, graphics accelerator 114 further includes a texture map memory structure 105, stored in special texture mapping memory 109, for holding data on which texture mapping is being performed by logic engine 101. Preferably memory 109 is configured in a 3D matrix for storing 3D representations both prior to and after texturing of such representations. Preferably, this memory 109 is relatively fast random access memory (RAM). In FIG. 1B, texture memory 109 is shown as a component of graphics accelerator 114. Alternatively, this memory 109 can be a portion (not shown) of main memory 104. Preferably, texture map 105 is a 3D structure, implemented in Cartesian coordinates. Alternatively, map 105 can be a 2D structure, with any required 3D operations implemented reading and writing from and to, respectively, other memory, such as main memory 104.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

---

☐ 8. Document ID: US 6545684 B1

L9: Entry 8 of 25 File: USPT Apr 8, 2003

DOCUMENT-IDENTIFIER: US 6545684 B1
TITLE: Accessing data stored in a memory

Detailed Description Text (24):
These pages may be contiguous portions of system memory or, alternatively, they may be discontiguous portions of system memory 21. In either case, process 37 allocates (605) the available portions of system memory for use by graphics processor 19. The available portions of memory are then tiled (606) in accordance with process 34 (FIG. 4). Following process 34, process 37 generates (207) a memory map to the tiles of system memory (and to graphics memory 22, if applicable). In this embodiment, the memory map is a page table that is generated by process 37 and programmed into cache 30 of graphics processor 19. The table itself may already exist in cache 30, in which case process 37 reprograms the table.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

---

9. Document ID: US 6518965 B2

L9: Entry 9 of 25 File: USPT Feb 11, 2003

DOCUMENT-IDENTIFIER: US 6518965 B2
TITLE: Graphics system and method for rendering independent 2D and 3D objects using pointer based display list video refresh operations

Detailed Description Text (225):
Graphics source data is preferably read into the on-chip cache memories for consumption by the Graphics Engines iteration and texture mapping units. In addition, the cache controllers for the Graphics Engine maintain a list of "in-Scope" primitives and manage the data reads and writes of the on-chip cache memories.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

---

10. Document ID: US 6483505 B1

L9: Entry 10 of 25 File: USPT Nov 19, 2002

DOCUMENT-IDENTIFIER: US 6483505 B1
TITLE: Method and apparatus for multipass pixel processing

CLAIMS:

12. The video graphics circuit of claim 11 further comprises a texture memory operably coupled to the texture mapping block of the graphics pipeline, wherein the texture memory stores at least one lookup table, wherein the at least one lookup table stores lookup values.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

---

11. Document ID: US 6072507 A

L9: Entry 11 of 25 File: USPT Jun 6, 2000

DOCUMENT-IDENTIFIER: US 6072507 A
TITLE: Method and apparatus for mapping a linear address to a tiled address

Detailed Description Text (12):
Having obtained the band pointer 52, the linear band offset 56, the normalized linear address 54, the tiled address 50 may be obtained. This is shown in the right portion of FIG. 4. As shown, the memory mapping is done on a tiled basis wherein at the initial address 50, the first tile is mapped in. This corresponds with the first 256 normalized linear addresses of the memory shown in FIG. 3. Referring back to FIG. 4, the next tile is mapped into memory in a zigzag fashion as shown. This zigzag fashion of mapping the tiles into memory continues until the actual tiled address 50 is reached. Once the tiled address is reached, a tiled band offset 58 may be obtained which is the difference between the band pointer 52 and the tiled address 50. As such, by following the graphical representation of FIG. 4, the linear address 55 can be mapped to the tiled address 50. Thus, when the central processing unit provides the linear address 55, the mapping circuit 28 of the video graphics processor may readily determine the tiled address 50. Note that by mapping the memory as shown in FIGS. 3 and 4, there is no latency from jumping from one tile to the next within the same band. Thus, the seven cycles to read a subsequent page is eliminated when the pages are associated with the same band. Further note that double bands may be included in the linearized mapping such that if the first two tiles of each band would be associated with each other thereby eliminating latency between jumping between two bands. By eliminating latency when retrieving pages of memory, the video graphics processing circuit saves 6 cycles each time page latency is avoided. As one can readily appreciate, when a 640.times.480 screen is being rendered, there is a substantial amount of time saved by eliminating such page latency.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments   KWIC | Draw Desc | Image

---

12. Document ID: US 6054993 A

L9: Entry 12 of 25 File: USPT Apr 25, 2000

DOCUMENT-IDENTIFIER: US 6054993 A
TITLE: Chroma-keyed specular texture mapping in a graphics processor

Detailed Description Text (6):
In accordance with the preferred embodiment, graphics processor 120 receives data in the form of a display list from the CPU 102 or system memory 104 via the PCI bus 112. The display list is stored in a register file in graphics processor 120 or memory (not shown) directly coupled to the graphics processor 120. The display list includes all information needed to draw a polygon. As explained in greater detail below, some of the values from the display list are used by the polygon engine 124 and some values are used by the texture map engine 126. It is assumed each polygon includes an upper or main triangle (such as triangle 47 in FIG. 1) abutting a lower or opposite triangle (such as triangle 49). The values in the display list include the data needed to render both upper and lower triangles. Table I below includes an exemplary display list identifying the values that are included in the list (first column of Table I) and the description of each value (second column). References to X and Y values refer to the x, y coordinates of pixels on the display (referred to as x, y pixel space). References to U and V values refer to the coordinates of texels in a texture map which are identified as u,v coordinates. The u, v coordinate system of a texture map is referred to as u, v texture map space. Further, references to SPEC values refer to specular highlighting and are used in accordance with a preferred embodiment of the invention described below.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments   KWIC | Draw Desc | Image

☐ 13. Document ID: US 6002411 A

L9: Entry 13 of 25 File: USPT Dec 14, 1999

DOCUMENT-IDENTIFIER: US 6002411 A
**** See image for Certificate of Correction ****
TITLE: Integrated video and memory controller with data processing and graphical processing capabilities

Brief Summary Text (12):
The integrated memory controller of the preferred embodiment includes a bus interface unit which couples through FIFO buffers to an execution engine. The execution engine includes compression and decompression engines as well as a texture mapping engine according to the present invention. The execution engine in turn couples to a graphics engine which couples through FIFO buffers to one or more symmetrical memory control units. The graphics engine is similar in function to graphics processors in conventional computer systems and includes line and triangle rendering operations as well as span line interpolators. An instruction storage/decode block is coupled to the bus interface logic which stores instructions for the graphics engine and memory compression/decompression engines. A Window Assembler is coupled to the one or more memory control units. The Window Assembler in turn couples to a display storage buffer and then to a display memory shifter. The display memory shifter couples to separate digital to analog converters (DACs) which provide the RGB signals and the synchronization signal outputs to the display monitor. The window assembler includes a novel display list-based method of assembling pixel data on the screen during screen refresh, thereby improving system performance. In addition, a novel anti-aliasing method is applied to the video data as the data is transferred from system memory to the display screen. The internal graphics pipeline of the IMC is optimized for high end 2D and 3D graphical display operations, as well as audio operations, and all data is subject to operation within the execution engine and/or the graphics engine as it travels through the data path of the IMC.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments   KWIC | Draw Desc | Image

☐ 14. Document ID: US 5999199 A

L9: Entry 14 of 25 File: USPT Dec 7, 1999

DOCUMENT-IDENTIFIER: US 5999199 A
TITLE: Non-sequential fetch and store of XY pixel data in a graphics processor

Brief Summary Text (10):
Storing the data and text in main memory may require a graphics drawing engine to access the texture maps via system bus external to the graphics processor. Each such access and transfer of the texture maps results in processing latency. These latencies substantially slow down the rate at which the graphics processor can therefore process pixel data.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments   KWIC | Draw Desc | Image

☐ 15. Document ID: US 5929869 A

L9: Entry 15 of 25 File: USPT Jul 27, 1999

DOCUMENT-IDENTIFIER: US 5929869 A
TITLE: Texture map storage with UV remapping

Brief Summary Text (9):
Texture maps in memory may also be stored in a local relatively fast and small "local" memory in a graphics device, or such maps may be stored in the relatively slow and large "systen" or "host" menory. When stored in the local memory, texture nap storage becomes a critical design concern and must be given special design consideration in order to naxinize the efficiency and the efficient use of the relatively expensive fast local memory. This storage priority and design consideration is further complicated since texture map information is generally stored relative to a "U-V" reference and local memory is usually organized with an "X-Y" addressing scheme. With a local XY memory, after higher priority storage space has been taken, there may be ample storage area in "X" space for a given WI texture map, but not enough dimension in the "Y" direction to store a designated texture map block. Thus, it becomes necessary to develop a local memory storage remapping scheme in order to reduce the effective "V" space required of a texture storage block.

Detailed Description Text (7):
In accordance with the present disclosure, texture map information is reconfigured from its normal "block" configuration as necessary in order to store as many texture maps as possible into available and unused sections of the local and relatively fast RDRAM frame buffer memory of the graphics subsystem rather than storing all of the texture map sections in the host or system memory. That local frame buffer storage will allow access and delivery of more texture maps to the graphics engine from local frame buffer memory rather than main host memory, and thereby increase the speed of the system significantly since fewer host memory accesses and arbitrations will be required during the operation of the graphics subsystem.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

---

☐ 16. Document ID: US 5852451 A

L9: Entry 16 of 25 File: USPT Dec 22, 1998

DOCUMENT-IDENTIFIER: US 5852451 A
**** See image for Certificate of Correction ****
TITLE: Pixel reordering for improved texture mapping

Abstract Text (1):
A system and method for reordering memory references for pixels to improved bandwidth and performance in texture mapping systems and other graphics systems by improving memory locality in conventional page-mode memory systems. Pixel memory references are received from a client graphics engine and placed in a pixel priority heap. The pixel priority heap reorders the pixel memory references so that references requiring a currently open page are, in general, processed before references that require page breaks. Reordered pixel memory references are transmitted to a memory controller for accessing memory.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

---

☐ 17. Document ID: US 5844576 A

L9: Entry 17 of 25 File: USPT Dec 1, 1998

DOCUMENT-IDENTIFIER: US 5844576 A
TITLE: Tiled linear host texture storage

Brief Summary Text (8):
Texture maps in memory may be stored in a local, relatively fast, RDRAM memory in a graphics device, or such maps may be stored in the system or host memory. When stored as part of a larger host memory, delays in accessing data from the host texture maps are encountered because of the speed of the memory type itself and also because of the nature of the configuration and access process of the host memory.

Brief Summary Text (9):
With specific reference to computer graphics applications, image texture information, such as color transparency of displayed images, is stored in texture maps. A texture map is a two dimensional array of "texels" consisting of "U" texels in the horizontal direction, and "V" lines of texels in the vertical direction. As a polygon is rendered, texels are fetched from a "texture map", processed for lighting and blending, and then such texels become "pixels" of the polygon. As an image is produced on a display screen, each line of data stored in a frame buffer is sequentially accessed and transferred to the display device to fill-in corresponding sequential lines of pixels on the display. The frame buffer is updated by a draw engine portion of the graphics system, which is, in turn, updated by a texture engine portion of the graphics system. The texture engine accesses a texture map which is usually stored in system or host memory. Each such access and transfer of texture information from host memory has a delay time associated therewith because of the inherent dependence of the storing and accessing process. For each access to a texture map stored in host memory, for example, there is a processing latency, as well as delays due to bus access and host memory access arbitration. Moreover, in graphics systems, the next texel of information needed is frequently not the next linear line of information from memory but rather the next texel in a different direction from the last texel transferred. Because of the nature of the linear storage of information in host memory, whenever a texel of information is required by the graphics engine which is not the next linearly displaced texel of information as stored in the host memory, then a new access to the host memory is required to locate the requested texel.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments   KWIC | Draw Desc | Image

---

☐ 18. Document ID: US 5828382 A

L9: Entry 18 of 25 File: USPT Oct 27, 1998

DOCUMENT-IDENTIFIER: US 5828382 A
TITLE: Apparatus for dynamic XY tiled texture caching

Brief Summary Text (9):
Most graphics subsystems store texture maps in main system memory. Storing the maps in main memory may require a graphics drawing engine to access the texture maps via a bus external to the graphics processor. Each such access and transfer of the texture maps results in processing delays due to inherent memory latency. These latencies substantially slow down the rate at which the graphics processor can therefore process the texture maps.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments   KWIC | Draw Desc | Image

---

☐ 19. Document ID: US 5790130 A

L9: Entry 19 of 25 File: USPT Aug 4, 1998

DOCUMENT-IDENTIFIER: US 5790130 A

**** See image for Certificate of Correction ****
TITLE: Texel cache interrupt daemon for virtual memory management of texture maps

Detailed Description Text (3):
The present invention relates to a software daemon that runs on the processor of a computer graphics system host computer and manages the storage of texture data within the local memory of a texture mapping graphics hardware device connected to the host computer. The daemon manages the local memory in such a way that the memory appears virtual. By contrast with prior art systems that download entire series of texture MIP maps into the local memory of the hardware device, the software manager of the present invention downloads only portions of texture MIP maps necessary at any one time to render an image or image portion. The manager considers the recent past frequency of use of particular texture map portions as well as the predicted future use of such map portions (based on relative priorities of the textures) in determining which portions to replace in the local hardware memory when new texture data is required by the hardware device to render an image or portion thereof.

CLAIMS:

22. A software daemon that manages texture data in a texture mapping computer graphics system, the system including a host computer having a system memory that stores texture data, a graphics hardware device, coupled to the host computer, that renders texture mapped images, and including a local memory that stores in blocks at least a portion of the texture data stored in the system memory at any one time, the blocks including portions of texture maps of larger-sized textures, the software daemon comprising:

a process that allocates the texture data among blocks for downloading to the hardware device;

a routine that determines which block is required by the hardware device during an interrupt;

a routine that selects the block within the hardware device to be replaced during the interrupt; and

a routine that downloads the required block to the hardware device to replace the selected block within the hardware device during the interrupt.

---

20. Document ID: US 5469535 A

L9: Entry 20 of 25 File: USPT Nov 21, 1995

DOCUMENT-IDENTIFIER: US 5469535 A
TITLE: Three-dimensional, texture mapping display system

Detailed Description Text (2):
Referring now in detail to the drawings, there is shown in FIG. 1 an overall block diagram of an improved three-dimensional, texture mapping display system 10 constructed in accordance with the principles of the present invention. The display system is used to display three-dimensional projected polygons with texture maps on a two-dimensional raster display device. The display system 10 is particularly adapted for use in video games to allow real-time animation of video game scenes with textured plane surfaces at a high speed of operation. The display system is comprised of a host computer 12 which is preferably a digital signal processor (DSP), a graphics co-processor 14, a texture memory 16, a video frame buffer 18, a color look-up table 20, and a cathode ray tube (CRT) or other display device 22.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

21. Document ID: JP 11149570 A

L9: Entry 21 of 25 File: JPAB Jun 2, 1999

DOCUMENT-IDENTIFIER: JP 11149570 A
TITLE: GAME DEVICE AND RECORDING MEDIUM USED FOR THE DEVICE

Abstract Text (2):
SOLUTION: A video memory 9 is provided with plural memory blocks 91, 92, and 93. A video decoder 7 decodes data obtained by encoding a two-dimensional picture to be used for texture mapping. Each memory block 91, 92, and 93 is respectively used as a memory block for allowing a three-dimensional graphic processor 10 to develop picture data, a memory block for allowing the three-dimensional graphic processor 10 to input a two-dimensional picture to be used for texture mapping, and a memory block for setting the two-dimensional picture decoded by the video decoder 7.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

22. Document ID: WO 9831004 A1

L9: Entry 22 of 25 File: EPAB Jul 16, 1998

PUB-NO: WO009831004A1
DOCUMENT-IDENTIFIER: WO 9831004 A1
TITLE: PIXEL REORDERING FOR IMPROVED TEXTURE MAPPING

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

23. Document ID: US 20030001857 A1

L9: Entry 23 of 25 File: DWPI Jan 2, 2003

DERWENT-ACC-NO: 2003-391782
DERWENT-WEEK: 200337

TITLE: Computer system includes graphics device which processes texture coordinates which are selectively routed from memory by mapping system

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Clip Img | Image

24. Document ID: JP 10021414 A

L9: Entry 24 of 25 File: DWPI Jan 23, 1998

DERWENT-ACC-NO: 1998-150533
DERWENT-WEEK: 199814

TITLE: Texture mapping circuit for computer graphics - has loading unit, which adds primitive pattern to memory block, based on comparison result of data read-out from register

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Clip Img | Image

---

☐ 25. Document ID: JP 09288742 A

L9: Entry 25 of 25 File: DWPI Nov 4, 1997

DERWENT-ACC-NO: 1998-029802
DERWENT-WEEK: 199803

TITLE: Parallel texture mapping system for 3D computer graphics - has texture read out unit which reads texture images from memory bank, based on generated co-ordinate values

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Clip Img | Image

---

Generate Collection | Print

Terms	Documents
L8 same memory	25

Display Format: - Change Format

Previous Page Next Page

WEST

Generate Collection | Print

Search Results - Record(s) 1 through 34 of 34 returned.

1. Document ID: US 20030095130 A1

L5: Entry 1 of 34 File: PGPB May 22, 2003

DOCUMENT-IDENTIFIER: US 20030095130 A1
TITLE: Texture mapping method

Summary of Invention Paragraph (5):
[0004] In a computer graphics system, a texture processing system can be illustrated as in FIG. 1. Various textures are stored in the texture storage device 10. The texture storage device 10 may be a hard drive. When a rendering engine in the computer graphics system requires one texture for texture mapping, the texture stored in the texture storage device 10 is sent to memory 12 of the graphics accelerator through a system bus 11. The rendering engine then employs texture processing unit 14 to read the texture from the memory 12 through memory bus 13, and performs a texture-mapping process.

Summary of Invention Paragraph (6):
[0005] Texture mapping is a very important technique for realism in computer generated 3D (three dimensional) images. Typically, a texture map is a two dimensional array of color values. The individual color values are called texels. Each texel has a unique address in the texture map. The address can be thought of as a column number and a row number, which are named u and v respectively. The 3D application can assign texture coordinates to any vertex of any primitive. When the computer graphics system renders a primitive, texture coordinates are calculated and the corresponding texels are accessed from memory for each pixel of the primitive. For instance, 3D applications can create objects that appear to have a wood grain pattern on them by mapping a wooden texture onto surfaces of 3D objects.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | Claims | KMC | Draw Desc | Image

2. Document ID: US 20030067473 A1

L5: Entry 2 of 34 File: PGPB Apr 10, 2003

DOCUMENT-IDENTIFIER: US 20030067473 A1
TITLE: Method and apparatus for executing a predefined instruction set

Summary of Invention Paragraph (4):
[0003] As is known, a conventional computing system includes a central processing unit, a chip set, system memory, a video graphics processor, and a display. The video graphics processor includes a raster engine and a frame buffer. The system memory includes geometric software and texture maps for processing video graphics data. The display may be a cathode ray tube (CRT) display, a liquid crystal display (LCD) or any other type of display. A typical prior art computing system of the type described above is illustrated in FIG. 1. As shown in FIG. 1, the system 100 includes a host 102 coupled to a graphics processor 104 and a display 106. The host 102 comprises the central processing unit, chip set and system memory as described above. The host 102 is responsible for the overall operation of the system 100. In particular, the host 102 provides, on a frame by frame

basis, video graphics data to the display 106 for display to a user of the system 100. The graphics processor 104, which comprises the raster engine and frame buffer, assists the host 102 in processing the video graphics data.

Detail Description Paragraph (6):
[0018] The vertex input memory 204 represents the data that is provided on a per vertex basis. In a preferred embodiment, there are sixteen vectors (a vector is a set of x, y, z and w coordinates) of input vertex memory available. During any single instruction cycle by the PVS engine 202, only a single operand is available from the vertex input memory 204. The constant memory 206 preferably comprises one hundred and ninety two vector locations for the storage of constant values. Likewise, only a single operand may be provided from the constant memory 206 to the PVS engine 202 during a single instruction cycle execution. The temporary register memory 208 is provided for the temporary storage of intermediate values calculated by the PVS engine 202. The temporary register memory 208 can provide no more than two input operands to the PVS engine 202 during a single clock cycle. In general, however, it is understood that the present invention is more broadly applicable to situations in which a data source is limited to n outputs but where instructions executed by the PVS engine 202 may require more than n inputs from the n-output source. To handle this situation, the PVS controller 214 of the present invention recognizes instructions of this type and provides at least two substitute instructions each of which requires no more than n operands from the n output source. This is described in greater detail with reference to FIG. 3.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | Claims | KWIC | Draw Desc | Image

3. Document ID: US 20030067472 A1

L5: Entry 3 of 34 File: PGPB Apr 10, 2003

DOCUMENT-IDENTIFIER: US 20030067472 A1
TITLE: Embedded memory system and method including data error correction

Detail Description Paragraph (5):
[0016] FIG. 2 illustrates circuitry included within the graphics processing system 132 for performing various graphics and video functions. As shown in FIG. 2, a bus interface-200 couples the graphics processing system 132 to the expansion bus 116 and optionally high-speed bus 136. In the case where the graphics processing system 132 is coupled to the processor 104 and the memory 108 through the high speed data bus 136 and the memory/bus interface 112, the bus interface 200 will include a DMA controller (not shown) to coordinate transfer of data to and from the host memory 108 and the processor 104. A graphics processor 204 is coupled to the bus interface 200 and is designed to perform various graphics and video processing functions, such as, but not limited to, generating vertex data and performing vertex transformations for polygon graphics primitives that are used to model 3D objects. The graphics processor 204 is coupled to a triangle engine 208 that includes circuitry for performing various graphics functions, such as clipping, attribute transformations, rendering of graphics primitives, and generating texture coordinates for a texture map.

Detail Description Paragraph (6):
[0017] A pixel engine 212 is coupled to receive the graphics data generated by the triangle engine 208. The pixel engine 212 contains circuitry for performing various graphics functions, such as, but not limited to, texture application or mapping, bilinear filtering, fog, blending, and color space conversion. A memory controller 216 coupled to the pixel engine 212 and the graphics processor 204 handles memory requests to and from a local memory 220. The local memory 220 stores graphics data, such as pixel values. A display controller 224 is coupled to the memory controller 216 to receive processed values for pixels that are to be displayed. The output values from the display controller 224 are subsequently provided to a display driver 232 that includes circuitry to provide digital signals, or convert digital signals to analog signals, to drive the display 140 (FIG. 1). It will be appreciated that the circuitry included in the graphics processing system 132 to practice embodiments of the present invention may be of conventional designs well understood by those of ordinary skill in the art.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | Claims | KWIC | Draw Desc | Image

4. Document ID: US 20030043159 A1

L5: Entry 4 of 34 File: PGPB Mar 6, 2003

DOCUMENT-IDENTIFIER: US 20030043159 A1
TITLE: Graphics resampling system and method for use thereof

Detail Description Paragraph (5):
[0018] FIG. 2 illustrates circuitry included within the graphics processing system 132 for performing various three-dimensional (3D) graphics functions. As shown in FIG. 2, a bus interface 200 couples the graphics processing system 132 to the expansion bus 116. In the case where the graphics processing system 132 is coupled to the processor 104 and the host memory 108 through the high speed data bus 136 and the memory/bus interface 112, the bus interface 200 will include a DMA controller (not shown) to coordinate transfer of data to and from the host memory 108 and the processor 104. A graphics processor 204 is coupled to the bus interface 200 and is designed to perform various graphics and video processing functions, such as, but not limited to, generating vertex data and performing vertex transformations for polygon graphics primitives that are used to model 3D objects. The graphics processor 204 is coupled to a triangle engine 208 that includes circuitry for performing various graphics functions, such as clipping, attribute transformations, rendering of graphics primitives, and generating texture coordinates for a texture map.

Detail Description Paragraph (6):
[0019] A pixel engine 212 is coupled to receive the graphics data generated by the triangle engine 208. The pixel engine 212 contains circuitry for performing various graphics functions, such as, but not limited to, texture application or mapping, bilinear filtering, fog, blending, and color space conversion. A memory controller 216 coupled to the pixel engine 212 and the graphics processor 204 handles memory requests to and from an local memory 220. The local memory 220 stores graphics data, such as source pixel color values and destination pixel color values. A display controller 224 is coupled to the memory controller 216 to receive processed destination color values for pixels that are to be rendered. Coupled to the display controller 224 is a resampling circuit 228 that facilitates resizing or resampling graphics images. As will be explained below, embodiments of the resampling circuit 228 perform approximations that simplify the calculation of a model between two sample points for use during resampling. The output color values from the resampling circuit 228 are subsequently provided to a display driver 232 that includes circuitry to provide digital color signals, or convert digital color signals to red, green, and blue analog color signals, to drive the display 140 (FIG. 1).

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments   KWIC | Draw Desc | Image

5. Document ID: US 20030030646 A1

L5: Entry 5 of 34 File: PGPB Feb 13, 2003

DOCUMENT-IDENTIFIER: US 20030030646 A1
TITLE: Trilinear texture filtering method with proper texel selection

Summary of Invention Paragraph (5):
[0004] In the processes of computer graphic systems, each pixel's color value has to be derived from texture by mapping, which is called texture filtering. In computer generated 3D images, texture filtering is a very important technique, Texture usually is a two-dimensional array of color values which are individually called texels. Each texel has its own address which is composed of a column and row numbers named S and T, respectively. Typically, each texel's texture coordinates (defined as "u" and "v") are in the range of 0.0 to 1.0. The coordinates also represent the texel center corresponding to a texture coordinate according to the texture mapping. In 3D applications, the texture

coordinates can be assigned to any vertex of any primitive. When the computer graphics system renders a primitive, texture coordinates are calculated and the corresponding texels are accessed from the memory for each pixel of the primitive. For instance, in 3D applications, an object that has a wood grain pattern can be created by mapping a wooden texture onto surfaces of the 3D object.

Summary of Invention Paragraph (6):
[0005] FIG. 1 is a texture processing in a computer graphic system, wherein the texture mapping data are stored in a texture storage device 10 such as a hard drive. When the rendering engine 12 requires the texture data for texture mapping, the texture is sent to the local memory 11 of the graphics accelerator through a system bus 13. Then, the rendering engine 12 reads the required texels from the local memory through the memory bus 14, and performs the texture-mapping processing. The existing texture-mapping methods include the nearest point sampling, bilinear texture filtering, and trilinear texture filtering.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments KWIC | Draw Desc | Image

6. Document ID: US 20030030643 A1

L5: Entry 6 of 34 File: PGPB Feb 13, 2003

DOCUMENT-IDENTIFIER: US 20030030643 A1
TITLE: Method and apparatus for updating state data

Summary of Invention Paragraph (4):
[0002] As is known, a conventional computing system includes a central processing unit, a chip set, system memory, a video graphics processor, and a display. The video graphics processor includes a raster engine and a frame buffer. The system or main memory includes geometric software and texture maps for processing video graphics data. The display may be a cathode ray tube (CRT) display, a liquid crystal display (LCD) or any other type of display. A typical prior art computing system of the type described above is illustrated in FIG. 1. As shown in FIG. 1, the system 100 includes a host 102 coupled to a graphics processor (or graphics processing circuit) 104 and main memory 108. The graphics processor 104 is coupled to local memory 110 and a display 106. The host 102 is responsible for the overall operation of the system 100. In particular, the host 102 provides, on a frame by frame basis, video graphics data to the display 106 for display to a user of the system 100. The graphics processor 104, which comprises the raster engine and frame buffer, assists the host 102 in processing the video graphics data. In a typical system, the graphics processor 104 processes three-dimensional (3D) processed pixels with host-created pixels in the local memory 110 of the graphics processor 104, and provides the combined result to the display 106.

Detail Description Paragraph (8):
[0017] The vertex input memory 204 represents the data that is provided on a per vertex basis. In a preferred embodiment, there are sixteen vectors (a vector is a set of x, y, z and w coordinates) of input vertex memory available. The constant memory 206 preferably comprises one hundred and ninety two vector locations for the storage of constant values. The temporary register memory 208 is provided for the temporary storage of intermediate values calculated by the PVS engine 202.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments KWIC | Draw Desc | Image

7. Document ID: US 20030001857 A1

L5: Entry 7 of 34 File: PGPB Jan 2, 2003

DOCUMENT-IDENTIFIER: US 20030001857 A1

TITLE: Method and apparatus for determining logical texture coordinate bindings

Abstract Paragraph (1):
A computer system and method are provided for mapping of texture images. This may include a memory device to store a plurality of texture coordinates associated with vertices of three dimensional objects and a graphics device coupled to the memory device to process internal texture coordinates. A mapping system may appropriately route select ones of the plurality of texture coordinates from the memory device to the graphics device. The texture images may be mapped onto objects that, when rendered, include the image that may later be displayed on a display device.

Detail Description Paragraph (16):
[0028] Embodiments of the present invention will be described with respect to a computer system that includes a memory device to store a plurality of texture coordinates associated with vertices of three dimensional objects, a graphics device to couple to the memory device and to process internal texture coordinates for display, and a mapping system to appropriately route select ones of the plurality of texture coordinates from the memory device to the graphics device.

Detail Description Paragraph (18):
[0030] During texture mapping, select information of the array 250 may be passed from the memory (i.e., the system memory 130) to a graphics device (i.e., the GMCH 140) to appropriately prepare the image. However, it may be disadvantageous to transfer the whole array 250 from the memory to the graphics device as the array 250 may contain an extremely large amount of data and may need to be reformatted at the graphics device. Additionally, hardware (i.e., the graphics device) may only be capable of storing a predetermined number of texture coordinates at one time. For example, the graphics device may only be capable of simultaneously storing data regarding four separate texture coordinates. Disadvantageous arrangements may require the use of software in the transfer of the array 250 to the graphics device. Some of the texture coordinates may be stripped from the array 250 after the array 250 has been transferred to the graphics device. This may involve the software copying the array 250 to an intermediate buffer and then passing select information in the intermediate buffer to the mapping engines. It is therefore desirable for a method and apparatus to transfer only select portions of the array 250 that will be used in the texture engines of the graphics device.

Detail Description Paragraph (19):
[0031] Embodiments of the present invention relate to a graphics device (such as the GMCH 140) supporting multiple texture mappings. A logical binding may be made between internal texture coordinate sets used by the device and externally-stored (i.e., within the system memory) vertex texture coordinates or a default value. The logical mapping may provide substantial flexibility with respect to the use, ordering and replication of vertex texture coordinates. Without this flexibility, the graphics driver may have to generate a second, rearranged copy of the vertex data at the costs of memory footprints, and additional processor overhead, complexity and thus lower system performance.

Detail Description Paragraph (21):
[0033] FIG. 5 illustrates texture coordinate data transfer according to an example embodiment of the present invention. Other embodiments and configurations are also within the scope of the present invention. More specifically, FIG. 5 illustrates a memory 260 (such as the system memory 130 of FIG. 3), a graphics device 270 (such as the GMCH 140 of FIG. 3) and software 280 that controls, among other things, the transfer of texture coordinates from the memory 260 to the graphics driver 270. The software 280 may reside in external memory. The memory 260 is shown as including the array 250. For illustration purposes, the array 250 may include texture coordinate data 252 and non-texture coordinate data 254. As show in FIG. 5, the graphics device 270 may include four texture mapping engines, namely a texture mapping engine (TME0) 272, a texture mapping engine (TME1) 274, a texture mapping engine (TME2) 276, and a texture mapping engine (TME3) 278. While this embodiment only illustrates four texture mapping engines within the graphics device 270, other numbers of texture mapping engines are also within the scope of the present invention.

Detail Description Paragraph (22):
[0034] The graphics device 270 may further include a register 271 associated with the texture mapping engine 272, a register 271 associated with the texture mapping engine 274, a register 275 associated with the texture mapping engine 276 and a register 277 associated with the texture mapping engine 278. Each of the registers 271, 273, 275 and 277 may be used to select the appropriate texture coordinate values to be obtained from the memory 260 (or a default value) for each of the texture mapping engine 272, the texture mapping engine 274, the texture mapping

engine 276 and the texture mapping engine 278, respectively. During operation, the software 280 may set values within the respective registers 271, 273, 275 and 277 such that the texture mapping engines 272, 274, 276 and 278 receive the appropriate texture coordinates from the memory 260. In accordance with embodiments of the present invention, the entire array 250 of texture coordinates does not needed to be transferred from the memory 260 to the graphics device 270. That is, embodiments of the present invention route the proper texture coordinates to the appropriate texture mapping engines and avoid transferring unneeded texture coordinates from the array 250. The software 280 may appropriately pick the texture coordinates to be transferred to the texture mapping engines 272, 274, 276 and 278. This avoids the graphics device 270 from having to reformat the vertex texture buffers after they have been transferred from the memory 260 to the graphics device 270.

Detail Description Paragraph (23):
[0035] FIGS. 6 and 7 illustrate how embodiments of the present invention may be useful to appropriately route the texture coordinate data from the memory 260 to the graphics device 270. As shown in FIG. 6, the memory 260 (of FIG. 5) may include eight vertex texture coordinates TC0, TC1, TC2, TC3, TC4, TC5, TC6 and TC7. These texture coordinates may be provided within the array 250 or may be provided within vertex texture buffers. The graphics device 270 (of FIG. 5) is represented in FIG. 6 as four internal texture coordinates (ITC0, ITC1, ITC2 and ITC3) that will be provided within the four texture mapping engines 272, 274, 276 and 278, respectively. In accordance with embodiments of the present invention, the four internal texture coordinates (ITC0, ITC1, ITC2 and ITC3) to be provided to the four texture mapping engines 272, 274, 276 and 278 (at the graphics device 270) may be default values 290 (such as 0,0,0) or may be one of the texture coordinates TC0, TC1, TC2, TC3, TC4, TC5, TC6 and TC7 provided within the memory 260. The selection as to which texture coordinates will be provided as the internal texture coordinates ITC0, ITC1, ITC2 and ITC3 (corresponding to the mapping engines 272, 274, 276 and 278) may be based on a signal TexCoordbinding 295. This signal may be provided by the software 280 to appropriately route the appropriate texture coordinates to the texture mapping engines 272, 274, 276 and 278. That is, the software 280 may specify, in this example, that the internal texture coordinates may come from any one of the texture coordinates TC0, TC1, TC2, TC3, TC4, TC5, TC6 and TC7 or from the default value. In other words, the software 280 may select the source of the texture coordinates for each texture mapping engine 272, 274, 276 and 278. The software 280 may make this selection based on the desired image to be created and the desire to avoid unneedlessly transferring data from the memory 260 to the graphics device 270. The apparatus appropriately routes (or transfers) the selected texture coordinates to the proper texture mapping engines 272, 274, 276 and 278 (as the internal texture coordinates ITC0, ITC1, ITC2 and ITC3) by utilizing the registers 271, 273, 275 and 277. Accordingly, the graphics device 270 may only obtain a limited number of texture coordinates from the memory 260 and avoid obtaining the unnecessary texture coordinates for a particular image. This additionally avoids the software 280 from having to reformat the array 250 when it arrives at the graphics device 270 as in disadvantageous arrangements.

Detail Description Paragraph (32):
[0044] Embodiments of the present invention may relate to a computer system that includes a memory device to store a plurality of texture coordinates associated with vertices of three dimensional objects, a graphics device having a plurality of mapping engines each to be used to map at least one of the objects based on a plurality of internal texture coordinates, and a mapping system to transfer select ones of the plurality of texture coordinates in the memory device to the mapping engines without transferring unselected one of the plurality of texture coordinates from the memory device to the mapping engines.

CLAIMS:

1. A computer system comprising: a memory device to store a plurality of texture coordinates associated with vertices of three dimensional objects; a graphics device to couple to said memory device and to process internal texture coordinates for display; and a mapping system to appropriately route select ones of said plurality of texture coordinates from said memory device to said graphics device.

8. A computer system comprising: a memory device to store a plurality of texture coordinates associated with vertices of three dimensional objects; a graphics device having a plurality of mapping engines each to map at least one of said objects based on a plurality of internal texture coordinates; and a mapping system to transfer select ones of said plurality of texture coordinates in said memory device to said mapping engines without transferring unselected ones of said plurality of texture coordinates from said memory device to said graphics device.

13. A graphics device for creating an image based on internal texture coordinates received from a memory device, said graphics device including a plurality of mapping engines and a plurality of registers, each register corresponding to a source of texture coordinate values for one of said mapping engines.

17. A method comprising: receiving a plurality of texture coordinate values in a memory device, said plurality of texture coordinates being associated with vertices of three dimensional objects; selecting ones of said plurality of texture coordinate values for mapping of at least one of said objects; and transferring said select ones of said plurality of texture coordinates values from said memory device to mapping engines.

22. A program storage device readable by machine, tangibly embodying a program of instructions executable by the machine to perform a method comprising: selecting ones of a plurality of texture coordinate values in a memory device, said plurality of texture coordinates values being associated with vertices of three dimensional objects; and transferring said select ones of said plurality of texture coordinates values from said memory device to mapping engines.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

8. Document ID: US 20020180749 A1

L5: Entry 8 of 34 File: PGPB Dec 5, 2002

DOCUMENT-IDENTIFIER: US 20020180749 A1
TITLE: Method of bilinear texture filtering

Summary of Invention Paragraph (4):
[0002] In the process of computer graphic system, each pixel's color value has to be derived from texture by mapping, which is called texture filtering. In computer generated 3D images, texture filtering is a very important technique. Texture usually is a two dimensional array of color values which are individually called texels. Each texel has own address which is formed with a column and row numbers named as S and T. Typically, each texel's texture coordinate is in the range of 0.0 to 1.0. The coordinates also represent texel center corresponding to a texture-mapping coordinate according to the texture mapping. In 3D applications, the texture coordinate can be assigned to any vertex of any primitive. When the computer graphics system renders a primitive, texture coordinates are calculated and the corresponding texels are accessed from memory for each pixel of the primitive. For instance, in 3D applications a object that has a wood grain pattern can be created by mapping a wooden texture onto surfaces of the 3D object.

Summary of Invention Paragraph (5):
[0003] FIG. 1 is the texture processing in computer graphic system, wherein the texture is stored in a texture storage unit 10 such as a hard drive. When a rendering engine requires the texture for texture mapping, the texture is sent to local memory II of the graphics accelerator through a bus. The texture processing unit 12 then reads the texture from the local memory 11 and performs texture-mapping process. The exiting texture mapping methods include nearest point sampling, bilinear texture filtering, and trilinear texture filtering.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

9. Document ID: US 20020140706 A1

L5: Entry 9 of 34 File: PGPB Oct 3, 2002

DOCUMENT-IDENTIFIER: US 20020140706 A1
TITLE: Multi-sample method and system for rendering antialiased images

Detail Description Paragraph (5):
[0024] FIG. 2 illustrates circuitry included within the graphics processing system 132 for performing various graphics and video functions. As shown in FIG. 2, a bus interface 200 couples the graphics processing system 132 to the expansion bus 116 and optionally high-speed bus 136. In the case where the graphics processing system 132 is coupled to the processor 104 and the memory 108 through the high speed data bus 136 and the memory/bus interface 112, the bus interface 200 will include a DMA controller (not shown) to coordinate transfer of data to and from the host memory 108 and the processor 104. A graphics processor 204 is coupled to the bus interface 200 and is designed to perform various graphics and video processing functions, such as, but not limited to, generating vertex data and performing vertex transformations for polygon graphics primitives that are used to model 3D objects. The graphics processor 204 is coupled to a triangle engine 208 that includes circuitry for performing various graphics functions, such as clipping, attribute transformations, rendering of graphics primitives, and generating texture coordinates for a texture map. The triangle engine further includes anti-aliasing circuitry 214 according to embodiments of the present invention. As will be described in more detail below, embodiments of the anti-aliasing circuitry 214 over-sample within the pixels of the graphics image to reduce aliasing artifacts.

Detail Description Paragraph (6):
[0025] A pixel engine 212 is coupled to receive the graphics data generated by the triangle engine 208. The pixel engine 212 contains circuitry for performing various graphics functions, such as, but not limited to, texture application or mapping, bilinear filtering, fog, blending, and color space conversion. A memory controller 216 coupled to the pixel engine 212 and the graphics processor 204 handles memory requests to and from a local memory 220. The local memory 220 stores graphics data, such as pixel values. A display controller 224 is coupled to the memory controller 216 to receive processed values for pixels that are to be displayed. The output values from the display controller 224 are subsequently provided to a display driver 232 that includes circuitry to provide digital signals, or convert digital signals to analog signals, to drive the display 140 (FIG. 1). It will be appreciated that the circuitry included in the graphics processing system 132 to practice embodiments of the present invention may be of conventional designs well understood by those of ordinary skill in the art.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

---

☐ 10. Document ID: US 20020140704 A1

L5: Entry 10 of 34 File: PGPB Oct 3, 2002

DOCUMENT-IDENTIFIER: US 20020140704 A1
TITLE: Resampling system and method for graphics data including sine-wave components

Detail Description Paragraph (5):
[0016] FIG. 2 illustrates circuitry included within the graphics processing system 132 for performing various three-dimensional (3D) graphics functions. As shown in FIG. 2, a bus interface 200 couples the graphics processing system 132 to the expansion bus 116. In the case where the graphics processing system 132 is coupled to the processor 104 and the host memory 108 through the high speed data bus 136 and the memory/bus interface 112, the bus interface 200 will include a DMA controller (not shown) to coordinate transfer of data to and from the host memory 108 and the processor 104. A graphics processor 204 is coupled to the bus interface 200 and is designed to perform various graphics and video processing functions, such as, but not limited to, generating vertex data and performing vertex transformations for polygon graphics primitives that are used to model 3D objects. The graphics processor 204 is coupled to a triangle engine 208 that includes circuitry for performing various graphics functions, such as clipping, attribute transformations, rendering of graphics primitives, and generating texture coordinates for a texture map.

Detail Description Paragraph (6):
[0017] A pixel engine 212 is coupled to receive the graphics data generated by the triangle engine 208. The pixel engine 212 contains circuitry for performing various graphics functions, such as, but not limited to, texture application or mapping, bilinear filtering, fog, blending, and color space conversion. A memory controller 216 coupled to the pixel engine 212 and the graphics processor 204 handles memory requests to and from an local

memory 220. The local memory 220 stores graphics data, such as source pixel color values and destination pixel color values. A display controller 224 is coupled to the memory controller 216 to receive processed destination color values for pixels that are to be rendered. Coupled to the display controller 224 is a resampling circuit 228 that facilitates resizing or resampling graphics images. As will be explained below, embodiments of the resampling circuit 228 perform approximations that simplify the calculation of a model between two sample points for use during resampling. The output color values from the resampling circuit 228 are subsequently provided to a display driver 232 that includes circuitry to provide digital color signals, or convert digital color signals to red, green, and blue analog color signals, to drive the display 140 (FIG. 1).

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

---

☐ 11. Document ID: US 20020136446 A1

L5: Entry 11 of 34 File: PGPB Sep 26, 2002

DOCUMENT-IDENTIFIER: US 20020136446 A1
TITLE: Pixel resampling system and method

Detail Description Paragraph (5):
[0016] FIG. 2 illustrates circuitry included within the graphics processing system 132 for performing various three-dimensional (3D) graphics functions. As shown in FIG. 2, a bus interface 200 couples the graphics processing system 132 to the expansion bus 116. In the case where the graphics processing system 132 is coupled to the processor 104 and the host memory 108 through the high speed data bus 136 and the memory/bus interface 112, the bus interface 200 will include a DMA controller (not shown) to coordinate transfer of data to and from the host memory 108 and the processor 104. A graphics processor 204 is coupled to the bus interface 200 and is designed to perform various graphics and video processing functions, such as, but not limited to, generating vertex data and performing vertex transformations for polygon graphics primitives that are used to model 3D objects. The graphics processor 204 is coupled to a triangle engine 208 that includes circuitry for performing various graphics functions, such as clipping, attribute transformations, rendering of graphics primitives, and generating texture coordinates for a texture map.

Detail Description Paragraph (6):
[0017] A pixel engine 212 is coupled to receive the graphics data generated by the triangle engine 208. The pixel engine 212 contains circuitry for performing various graphics functions, such as, but not limited to, texture application or mapping, bilinear filtering, fog, blending, and color space conversion. A memory controller 216 coupled to the pixel engine 212 and the graphics processor 204 handles memory requests to and from an local memory 220. The local memory 220 stores graphics data, such as source pixel color values and destination pixel color values. A display controller 224 is coupled to the memory controller 216 to receive processed destination color values for pixels that are to be rendered. Coupled to the display controller 224 is a resampling circuit 228 that facilitates resizing or resampling graphics images. As will be explained below, embodiments of the resampling circuit 228 perform approximations that simplify the calculation of a model between two sample points for use during resampling. The output color values from the resampling circuit 228 are subsequently provided to a display driver 232 that includes circuitry to provide digital color signals, or convert digital color signals to red, green, and blue analog color signals, to drive the display 140 (FIG. 1).

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

---

☐ 12. Document ID: US 20020135592 A1

L5: Entry 12 of 34 File: PGPB Sep 26, 2002

DOCUMENT-IDENTIFIER: US 20020135592 A1
TITLE: Pixel resampling system and method for text

Detail Description Paragraph (5):
[0021] FIG. 2 illustrates circuitry included within the graphics processing system 132 for performing various three-dimensional (3D) graphics functions. As shown in FIG. 2, a bus interface 200 couples the graphics processing system 132 to the expansion bus 116. In the case where the graphics processing system 132 is coupled to the processor 104 and the host memory 108 through the high speed data bus 136 and the memory/bus interface 112, the bus interface 200 will include a DMA controller (not shown) to coordinate transfer of data to and from the host memory 108 and the processor 104. A graphics processor 204 is coupled to the bus interface 200 and is designed to perform various graphics and video processing functions, such as, but not limited to, generating vertex data and performing vertex transformations for polygon graphics primitives that are used to model 3D objects. The graphics processor 204 is coupled to a triangle engine 208 that includes circuitry for performing various graphics functions, such as clipping, attribute transformations, rendering of graphics primitives, and generating texture coordinates for a texture map.

Detail Description Paragraph (6):
[0022] A pixel engine 212 is coupled to receive the graphics data generated by the triangle engine 208. The pixel engine 212 contains circuitry for performing various graphics functions, such as, but not limited to, texture application or mapping, bilinear filtering, fog, blending, and color space conversion. A memory controller 216 coupled to the pixel engine 212 and the graphics processor 204 handles memory requests to and from an local memory 220. The local memory 220 stores graphics data, such as source pixel color values and destination pixel color values. A display controller 224 is coupled to the memory controller 216 to receive processed destination color values for pixels that are to be rendered. Coupled to the display controller 224 is a resampling circuit 228 that facilitates resizing or resampling graphics images. As will be explained below, embodiments of the resampling circuit 228 perform approximations that simplify the calculation of a model between two sample points for use during resampling. The output color values from the resampling circuit 228 are subsequently provided to a display driver 232 that includes circuitry to provide digital color signals, or convert digital color signals to red, green, and blue analog color signals, to drive the display 140 (FIG. 1).

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

13. Document ID: US 20020107070 A1

L5: Entry 13 of 34 File: PGPB Aug 8, 2002

DOCUMENT-IDENTIFIER: US 20020107070 A1
TITLE: System and method for creating real-time shadows of complex transparent objects

Abstract Paragraph (1):
A system and method for creating real-time shadows of complex transparent objects includes a processor and a main memory that stores a transparent blocker object and a receiver object. A light source in a three-dimensional game environment determines an origin of a light coordinate system. A unit vector from the light source to the blocker object defines a z-axis of the light coordinate system and defines a light vector. The processor converts vertices of the blocker object from world coordinates into light coordinates. A graphics processor then calculates a shadow map by taking the dot-product of the light vector and each vertex of the blocker object. The shadow map is then stored in a memory of a graphics processor and is applied as a texture map to the receiver object by the graphics processor.

Summary of Invention Paragraph (10):
[0008] The shadow map is calculated by determining a light vector and a light coordinate system, converting vertices of the transparent object into light coordinates, and taking a dot product of the light vector and each vertex of the transparent object in light coordinates. The shadow map is stored in a memory of the graphics processor, and

then applied to the receiver object as a texture map. The graphics processor, in conjunction with the vector processing unit, preferably applies the shadow map to the receiver object using multiplicative pixel-blending techniques.

Detail Description Paragraph (14):
[0030] The vertices of transparent object 214 are then converted into light coordinates. The vertices of transparent object 214 are typically expressed as three-dimensional points in world coordinates in main memory 110. CPU 112 transforms the vertices of transparent object 214 from world coordinates into light coordinates using one or more conversion matrices.

CLAIMS:

2. The electronic entertainment system of claim 1, wherein the transparent object is stored in the memory as vertices that determine polygons in a world coordinate system.

10. The electronic entertainment system of claim 1, wherein the receiver object is stored in the memory as vertices that determine polygons in a world coordinate system.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments   KWIC | Draw Desc | Image

14. Document ID: US 20020081030 A1

L5: Entry 14 of 34 File: PGPB Jun 27, 2002

DOCUMENT-IDENTIFIER: US 20020081030 A1
TITLE: System and method for detecting text in mixed graphics data

Detail Description Paragraph (5):
[0016] FIG. 2 illustrates circuitry included within the graphics processing system 132 for performing various three-dimensional (3D) graphics functions. As shown in FIG. 2, a bus interface 200 couples the graphics processing system 132 to the expansion bus 116. In the case where the graphics processing system 132 is coupled to the processor 104 and the host memory 108 through the high speed data bus 136 and the memory/bus interface 112, the bus interface 200 will include a DMA controller (not shown) to coordinate transfer of data to and from the host memory 108 and the processor 104. A graphics processor 204 is coupled to the bus interface 200 and is designed to perform various graphics and video processing functions, such as, but not limited to, generating vertex data and performing vertex transformations for polygon graphics primitives that are used to model 3D objects. The graphics processor 204 is coupled to a triangle engine 208 that includes circuitry for performing various graphics functions, such as clipping, attribute transformations, rendering of graphics primitives, and generating texture coordinates for a texture map.

Detail Description Paragraph (6):
[0017] A pixel engine 212 is coupled to receive the graphics data generated by the triangle engine 208. The pixel engine 212 contains circuitry for performing various graphics functions, such as, but not limited to, texture application or mapping, bilinear filtering, fog, blending, and color space conversion. A memory controller 216 coupled to the pixel engine 212 and the graphics processor 204 handles memory requests to and from an local memory 220. The local memory 220 stores graphics data, such as source pixel color values and destination pixel color values. A display controller 224 is coupled to the memory controller 216 to receive processed destination color values for pixels that are to be rendered. Coupled to the display controller 224 is a resampling circuit 228 that facilitates resizing or resampling graphics images. As will be explained below, embodiments of the resampling circuit 228 according to embodiments of the present invention include a text detection circuit which identifies whether a sampling of pixels includes graphics data that represents text images. The output color values from the resampling circuit 228 are subsequently provided to a display driver 232 that includes circuitry to provide digital color signals, or convert digital color signals to red, green, and blue analog color signals, to drive the display 140 (FIG. 1).

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

☐ 15. Document ID: US 20020070941 A1

L5: Entry 15 of 34 File: PGPB Jun 13, 2002

DOCUMENT-IDENTIFIER: US 20020070941 A1
TITLE: Memory system having programmable multiple and continuous memory regions and method of use thereof

Summary of Invention Paragraph (5):
[0003] Generally, embedded memory included in a graphics processing system allows data to be provided to processing circuits, such as the graphics processor, the pixel engine, and the like, with low access times. The proximity of the embedded memory to the graphics processor and its dedicated purpose of storing data related to the processing of graphics information enable data to be moved throughout the graphics processing system quickly. Thus, the processing elements of the graphics processing system may retrieve, process, and provide graphics data quickly and efficiently, increasing the processing throughput. The embedded memory is used by the graphics processing system for a variety of purposes. For example, the embedded memory is often allocated for z-buffering purposes to store the depth values of graphics primitives in a three-dimensional image. Another use is as a pixel buffer to store the color values of pixels that are used for processing, or that will be rendered. Still another use is as a texture buffer to store texture map data where texture mapping is to be applied during rendering a graphics image. By allocating the embedded memory for these purposes, the overall processing speed of the graphics processing system is increased.

Detail Description Paragraph (6):
[0018] FIG. 2 illustrates circuitry included within the graphics processing system 132 for performing various three-dimensional (3D) graphics functions. As shown in FIG. 2, a bus interface 200 couples the graphics processing system 132 to the expansion bus 116. In the case where the graphics processing system 132 is coupled to the processor 104 and the host memory 108 through the high speed data bus 136 and the memory/bus interface 112, the bus interface 200 will include a DMA controller (not shown) to coordinate transfer of data to and from the host memory 108 and the processor 104. A graphics processor 204 is coupled to the bus interface 200 and is designed to perform various graphics and video processing functions, such as, but not limited to, generating vertex data and performing vertex transformations for polygon graphics primitives that are used to model 3D objects. The graphics processor 204 is coupled to a triangle engine 208 that includes circuitry for performing various graphics functions, such as clipping, attribute transformations, rendering of graphics primitives, and generating texture coordinates for a texture map. A pixel engine 212 is coupled to receive the graphics data generated by the triangle engine 208. The pixel engine 212 contains circuitry for performing various graphics functions, such as, but not limited to, texture application or mapping, bilinear filtering, fog, blending, and color space conversion.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

☐ 16. Document ID: US 6597363 B1

L5: Entry 16 of 34 File: USPT Jul 22, 2003

DOCUMENT-IDENTIFIER: US 6597363 B1
TITLE: Graphics processor with deferred shading

Brief Summary Text (182):
When a page in sort memory is being written, vertices and modes are written sequentially into the sort memory as they are received by the sort block. When a page is read from sort memory, the read is done on a tile-by-tile basis, and the read process operates in two modes: (1) time order mode, and (2) sorted transparency mode.

Detailed Description Text (98):
The GEO block is the first computation unit at the front end of the graphical pipeline. It deals mainly with per-vertex operations, like the transformation of vertex coordinates and normals. The Frontend (i.e., AGI and CFD Blocks) deals with fetching and decoding the Graphics Hardware Commands. The Frontend loads the necessary transform matrices, material and light parameters and other mode settings into the input registers of the GEO block. The GEO block sends transformed vertex coordinates, normals, generated and/or transformed texture coordinates, and per-vertex colors, to the Mode Extraction and Sort blocks. Mode Extraction stores the "color" data and modes in the Polygon memory. Sort organizes the per-vertex "spatial" data by Tile and writes it into the Sort Memory.

Detailed Description Text (186):
MEX receives a sequence of packets from GEO. For each primitive, MEX first receives the relevant state packets and then it receives the geometry packets. (Color vertex information is received before the sort vertex information.) The sort vertex data consists of the information needed for sorting and culling of primitives such as the clipped window coordinates. The VtxMode packet contains information about depth test etc. The information in CullMode, VtxMode and sort vertex packets is sent to the Sort-Setup-Cull part of the pipeline. The "color" vertex data consists of information needed for lighting and texturing of primitive fragments such as the vertex eye-coordinates, vertex normals, texture coordinates etc and is saved in polygon memory to be retrieved later.

Detailed Description Text (223):
As described in the chapter "Architectural Overvievw", the window (the display area on the screen) is divided horizontally and vertically into a set of tiles, and Sort keeps an ordered list for each tile. When a page of Sort Memory is being written, vertices and modes are written sequentially into the Sort Memory as they are received by the Sort Block. When a page of Sort Memory is read, it is done on a tile-by-tile basis. The read process operates in two modes: 1) Time Order Mode; and 2) Sorted Transparency Mode. In Time Order Mode, time order of vertices and modes are preserved within each tile. That is, for a given tile, vertices and modes are read in the same order as they are written. In Sorted Transparency Mode, reading of each tile is divided into multiple passes, where, in the first pass, guaranteed opaque geometry is output from the Sort Block, and, in subsequent passes, potentially transparent geometry is output from the Sort Block. Within each pass, the time ordering is preserved, and mode data is inserted in its correct time-order location.

Detailed Description Text (227):
FIG. 31 shows a simple example of data stored into a Sort Memory Page, including only six tiles and eight primitives. As seen the example, each Sort Memory Page is divided in two: 1) Data Storage; and 2) Pointer Storage. Data Storage stores its data in the received order, and stores two types of Storage Entries: 1) vertex packets and 2) mode packets. The example in FIG. 31 shows thirteen (13) vertex packets and three mode packets. Pointer Storage contains two types of lists: 1) Tile Pointer Lists, one for each tile; and 2) a Mode Pointer List. The example in FIG. 31 shows six Tile Pointer Lists containing a total of 18 Vertex Pointers, and also shows the Mode Pointer List containing a Clear Pointer and three Cull Pointers. The size of vertex packets and mode packets is always a single Rambus Dualoct. Hence, the addresses shown in FIG. 31 are Dualoct addresses.

Detailed Description Text (233):
During the reading of a Sort Memory Page, multiple vertices are assembled into Sort Primitives, and modes are injected into the output stream of every tile such that Sort Memory addresses are always increasing. In the example of FIG. 31, when Tile 3 is read, the order of output packets is (specified as Data Storage addresses): 0, 1, 4, 5, 6, 7, 12, and 13. Notice only the primitives that touch Tile 3 (i.e., 4, 5, 6, and 12) are output, but all four mode packets are output. In general, all mode packets are sent during every tile, however, some mode packets are not fed to the output stream because they can be eliminated or are not relevant for the target draw buffer. This is discussed in detail in a later section.

Detailed Description Text (463):
The Cull Unit receives Setup Output Primitive Packets that each describe, on a per tile basis, either a triangle, a line or a point. Sort is the unit that bins the incoming geometry entities to tiles. Setup is the unit that pre-processed the primitives to provide more detailed geometric information for Cull to do the hidden surface removal. Setup will pre-calculate the slope value for all the edges, the bounding box of the primitive within the tile, minimum depth value (front most) of the primitive within the tile, and other relevant data. Prior to Sort, Mode Extraction has already

extracted the information of color, light, texture and related mode data, Cull only gets the mode data that is relevant to Cull and a pointer, called Color Pointer, that points to color, light and texture data stored in Polygon Memory.

CLAIMS:

1. A graphics rendering system for forming a rendered image from graphics primitives, the rendered image stored in a frame buffer, the graphics rendering device comprising: (A) a host processor; (B) a system memory; (C) a frame buffer storing pixels; (D) a device for receiving commands comprising: (1) a 2D command queue memory receiving and storing first direct memory access commands and first commands to perform two-dimensional graphics operations; (2) a 3D command queue memory receiving and storing second direct memory access commands and second commands to perform three-dimensional graphics operations, the second commands comprising: some of the graphics primitives; and part of a pipeline state; (3) a 2D response queue memory receiving and storing third commands to perform two-dimensional graphics operations; (4) a 3D response queue memory receiving and storing fourth commands to perform three-dimensional graphics operations, the fourth commands comprising: some of the graphics primitives; and part of the pipeline state; and (5) a direct memory access controller coupled to the 2D command queue memory and the 3D command queue memory, the direct memory access controller comprising: (a) logic receiving the stored first direct memory access commands and performing the first direct memory access commands by reading the third commands from the system memory and writing the read third commands into the 2D response queue; and (b) logic receiving the stored second direct memory access commands and performing the second direct memory access commands by reading the fourth commands from the system memory and writing the read fourth commands into the 3D response queue. (E) a geometry unit comprising: (1) logic transforming the graphics primitives; (2) logic clip testing the graphics primitives to determine if the graphics primitives are at least partially contained in a view volume; (3) logic performing face determination to discard any of the graphics primitives that are facing away from a viewing point; (F) a sort memory storing: (1) the part of the pipeline state needed for hidden surface removal; and (2) the part of the graphics primitives needed for hidden surface removal, comprising vertex locations; (G) a state memory storing: (1) the graphics primitives for a frame with attributes of the graphics primitives; and (2) the part of the graphics primitives not needed for hidden surface removal, comprising vertex colors and texture coordinates; (H) a sort unit comprising: (1) logic sorting the graphics primitives according to tile areas within the image; (2) logic reading the sorted graphics primitives from the sort memory according to tile; (I) a setup unit computing derivatives for the sorted graphics primitives; (J) a cull unit performing hidden surface removal to determine which portions of the graphics primitives stored in the scene memory affect the final color of the pixels in the frame buffer; (K) one or more computational units generating fragment color for each sample or group of samples within each of the pixels, the generation of color values for each of the pixels being done only for the portions of the graphics primitives that are determined to affect the pixels, the computational units comprising: (1) a device for interpolating data associated with the graphics primitives, the device comprising: (a) a memory storing cache fill polygon color data, the color data comprising vertex coordinates, w coordinates, and texture coordinates; (b) logic computing barycentric coefficients for each of the graphics primitives from the vertex coordinates and the w coordinates; (c) a cache memory storing a plurality of data cache entries corresponding to a plurality of the graphics primitives, the cached data comprising barycentric coefficients and texture coordinates; (d) a register receiving a fragment of one of the graphics primitives, the fragment comprising fragment coordinates and a sample mask, the sample mask indicating sample locations within the pixel covered by the fragment; (e) a lookup table determining lower bits of an interpolation location, an input of the lookup table comprising the sample mask, the interpolation location having sub-pixel accuracy; (f) logic selecting and reading one of the stored data cache entries corresponding to the received fragment; (g) logic computing interpolation coefficients from the barycentric coefficients and the interpolation location; and (h) logic interpolating the texture coordinates using the computed interpolation coefficients, generating texture coordinates at the interpolation location; (2) a device for interpolating vertex surface normals comprising: (a) logic decomposing each of the vertex surface normals into a vertex magnitude vector and a vertex direction vector; (b) logic computing an interpolated magnitude vector from a plurality of the vertex magnitude vectors and the computed interpolation coefficients; (c) logic computing an interpolated direction vector from a plurality of the vertex direction vectors and the computed interpolation coefficients; and (d) logic combining the interpolated magnitude vector and the interpolated direction vector to generate an interpolated surface normal. (L) a pixel unit comprising: (1) logic blending the fragment colors together to generate per-pixel color values; and (2) logic storing the generated per-pixel color values into the frame buffer; (M) a status monitoring device comprising: (1) logic associating object tags with the three-dimensional objects as the three-dimensional objects are processed by the graphics rendering system; (2) logic testing the three-dimensional objects with one or more visibility criteria as the three-dimensional objects are processed in the graphics rendering system; (3) a status register comprising a plurality of bits; (4) logic selecting one of the bits in the status register according to one of the object tags; (5) logic setting the value of the selected bit in the status register according to response of the corresponding one of the three-dimensional objects to the testing with the visibility criteria; and (6) logic reading and transmitting the status register contents to the host

processor, the host processor determining which of the three-dimensional objects are at least partially visible; (N) a device for storing a digital image into the frame buffer comprising: (1) a first memory storing a window identifier map, the window identifier map comprising a window identifier for each of the pixels in the frame buffer; (2) a second memory storing a plurality of bounding box descriptions, each of the bounding box descriptions comprising maximum and minimum pixel coordinates of the corresponding bounding box and a bounding box window identifier; (3) a register storing a programmatically assigned image identifier associated with the digital image; (4) logic selecting an image pixel from the digital image; (5) pixel ownership logic operating on the selected image pixel, the pixel ownership logic comprising: (a) window identifier map test logic enabled by a programmatically set control bit, the window identifier map test logic comprising: (i) logic reading the window identifier from the first memory that corresponds to the selected image pixel; (ii) a comparator circuit comparing the assigned image identifier to the read window identifier; and (iii) logic discarding the selected image pixel if the assigned image identifier does not match the read window identifier; and (b) bounding box test logic enabled by a programmatically set control bit, the bounding box test logic comprising: (i) logic selecting a first one of the bounding box descriptions such that the selected pixel is within the maximum and minimum pixel coordinates of the selected first bounding box description; (ii) a comparator circuit comparing the assigned image identifier to the bounding box window identifier of the first bounding box description; and (iii) logic discarding the selected image pixel if the assigned image identifier does not match the bounding box window identifier of the first bounding box; and (c) logic writing the selected image pixel to the frame buffer if the selected image pixel was not discarded. (O) a scanout device comprising: (1) logic generating a histogram of the pixel values in the rendered image as the pixel values are stored into the frame buffer; (2) logic computing a transformation function, based on the histogram, to adjust the scale and dynamic range of the pixel values so as to match the digital-to-analog converter; (3) logic reading the pixel values from the frame buffer in a raster line order; (4) logic transforming the read pixel values according to the computed transformation function; and (5) logic inputting the transformed pixel values to the digital-to-analog converter. (P) a scanout device comprising: (1) a memory storing programmatically downloaded interpolation coefficients; (2) a register storing a programmatically selected one of a plurality of fixed zoom ratios; (3) a register storing a programmatically selected location of a zoom bounding box within the computer display; (4) a register storing a programmatically assigned a window identifier associated with the zoom bounding box; (5) logic reading a first pixel color from the frame buffer; (6) logic reading a plurality of second pixel colors from the frame buffer; (7) logic computing a zoomed pixel color from the plurality of second pixel colors and the downloaded interpolation coefficients; (8) logic reading a pixel window identifier from the frame buffer; (9) logic sending the computed zoomed pixel color to the digital-to-analog converter if the display pixel is within the zoom bounding box and the pixel window identifier equals the window identifier assigned to the zoom bounding box; and (10) logic sending the read first pixel color to the digital-to-analog converter if the display pixel is not within the zoom bounding box or the pixel window identifier does not equal the window identifier assigned to the zoom bounding box.

5. A graphics rendering method for forming a rendered image stored in a frame buffer, the graphics rendering method comprising the steps: receiving graphics primitives; transforming the graphics primitives; clip testing the graphics primitives to determine if the graphics primitives are at least partially contained in a view volume; face determination to discard any of the graphics primitives that are facing away from a viewing point; sorting the graphics primitives according to tile areas within the image; storing, in a sort memory: (1) the part of the pipeline state needed for hidden surface removal; and (2) the part of the graphics primitives needed for hidden surface removal, comprising vertex locations; storing, in a state memory: (1) the part of the pipeline state not needed for hidden surface removal; and (2) the part of the graphics primitives not needed for hidden surface removal, comprising vertex colors and texture coordinates; reading the sorted graphics primitives from the sort memory according to tile; performing setup for incremental render on the sorted graphics primitives read from the sort memory, comprising the computing of derivatives; hidden surface removal to determine which portions of the sorted graphics primitives affect the final color of pixels in the frame buffer; fragment coloring to generate color values for each fragment, each fragment comprising a sample or a group of samples within one of the pixels, the generation of color values for each of the fragments being done only for the portions of the graphics primitives that are determined to affect the final color of pixels in the frame buffer; blending the fragment colors together to generate a single color value per pixel; and storing the generated per-pixel color values into the frame buffer.

20. A graphics rendering device for forming a rendered image from graphics primitives, the rendered image stored in a frame buffer, the graphics rendering device comprising: (a) a geometry unit comprising: (1) logic transforming the graphics primitives; (2) logic clip testing the graphics primitives to determine if the graphics primitives are at least partially contained in the view volume; (3) logic performing face determination to discard any of the graphics primitives that are facing away from the viewing point; (b) a sort memory storing: (1) the part of the pipeline state needed for hidden surface removal; and (2) the part of the graphics primitives needed for hidden surface removal, comprising vertex locations; (c) a state memory storing: (1) the graphics primitives for a frame with attributes of the graphics primitives; and (2) the part of the graphics primitives not needed for hidden surface removal, comprising

vertex colors and texture coordinates; (d) a sort unit comprising: (1) logic sorting the graphics primitives according to tile areas within the image; (2) logic reading the sorted graphics primitives from the sort memory according to tile; (e) a setup unit computing derivatives for the sorted graphics primitives; (f) a cull unit performing hidden surface removal to determine which portions of the graphics primitives stored in the scene memory affect the final color of pixels in the frame buffer; (g) one or more computational units generating fragment color values for each sample or group of samples within each of the pixels, the generation of color values for each of the pixels being done only for the portions of the graphics primitives that are determined to affect the pixels; (h) a pixel unit comprising: (1) logic blending the fragment colors together to generate per-pixel color values; and (2) logic storing the generated per-pixel color values into the frame buffer.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments   KWIC | Draw Desc | Image

---

☐ 17. Document ID: US 6590579 B1

L5: Entry 17 of 34 File: USPT Jul 8, 2003

DOCUMENT-IDENTIFIER: US 6590579 B1
TITLE: System for low miss rate replacement of texture cache lines

Abstract Text (1):
A system and method is provided for mipmap texturing in which texture tiles are mapped into sets of a set-associative texture cache for use in displaying a graphic primitive. When a miss occurs, a new texture tile is called from main memory to replace a texture tile which is not shared between the segment being traversed and the next segment to be traversed and which is the "least recently used". This is accomplished by maintaining a record for each cache line describing the texture tile it contains and replacing the texture tile which is the "least likely to be reused".

CLAIMS:

1. Apparatus comprising: a memory containing a plurality of texture tiles and an output having a plurality of pixels capable of being traversed in horizontal bands and vertical columns defining a plurality of segments; said output capable of having a primitive displayed thereon and a texture displayed on said primitive; a cache containing a subset of said plurality of texture tiles; a graphics processor connected to said cache and said memory, said processor using said texture tile to map said texture on said plurality of pixels on said output, said processor using a texture tile from said subset when a processor required texture tile is in said cache and from said plurality of texture tiles in said memory when said processor required texture tile is not in said cache; a plurality of registers in said cache, each of said registers associated with each of said plurality of texture tiles in said subset of said plurality of texture tiles in said cache, said register having an entry recording when a texture tile in a first segment is in a spatial position to be reused in a second traverse; and said graphics processor replaces a texture tile in said cache using said entries in said register when said processor uses a texture tile from said memory with said texture tile from said memory based on replacing the least likely texture tile to be reused of said texture tiles in said cache.

7. A cache texture tile replacement method comprising the steps of: traversing an output in a memory having a plurality of pixels in horizontal bands and vertical columns defining a plurality of segments; displaying a primitive on said output and a texture on said primitive; placing a plurality of texture tiles in a memory; placing a subset of said plurality of texture tiles in a cache; using said texture tile to map texture on to said plurality of pixels on said output with a graphics processor connected to said cache and said memory; using a texture tile from said subset when a processor required texture tile is in said cache; using a texture tile from said plurality of texture tiles in said memory when said processor required texture tile is not in said cache; recording an entry in a register associated with each of said plurality of texture tiles in said subset of said plurality of texture tiles in said cache when a texture tile in a first segment is in a spatial position to be reused in a second traverse; and replacing the least likely texture tile to be reused of said texture tiles in said cache using said entries in said register when said processor uses a texture tile from said memory to replace said texture tile in said cache.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

☐ 18. Document ID: US 6567084 B1

L5: Entry 18 of 34 File: USPT May 20, 2003

DOCUMENT-IDENTIFIER: US 6567084 B1
TITLE: Lighting effect computation circuit and method therefore

Brief Summary Text (4):
As is known, a conventional computing system includes a central processing unit, a chip set, system memory, a video graphics circuit, and a display. The video graphics circuit includes a raster engine and a frame buffer. The system memory includes geometric software and texture maps for processing video graphics data. The display may be a CRT display, LCD display or any other type of display.

Detailed Description Text (128):
The vertex distribution block 112 receives input vertex data 134, or at least portions thereof, and distributes this data on a vertex-by-vertex basis to the transform threads 114, 116, and 118. The distribution performed by the vertex distribution block 112 may be performed such that when a transform thread has completed processing a vertex, the vertex distribution block provides it with the next pending vertex to be processed. The input vertex data 134, or at least portions thereof, is also received by the input controller 62 such that data relating to the input vertex data to be processed by the various threads in the system will be available in the memory structures included in the circuit. The input vertex data stored in the memory structures may include spatial coordinates, color components, texture coordinates, and lighting effect parameters.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

☐ 19. Document ID: US 6556199 B1

L5: Entry 19 of 34 File: USPT Apr 29, 2003

DOCUMENT-IDENTIFIER: US 6556199 B1
TITLE: Method and apparatus for fast voxelization of volumetric models

Detailed Description Text (24):
Preferably, graphics accelerator 114 further includes a texture map memory structure 105, stored in special texture mapping memory 109, for holding data on which texture mapping is being performed by logic engine 101. Preferably memory 109 is configured in a 3D matrix for storing 3D representations both prior to and after texturing of such representations. Preferably, this memory 109 is relatively fast random access memory (RAM). In FIG. 1B, texture memory 109 is shown as a component of graphics accelerator 114. Alternatively, this memory 109 can be a portion (not shown) of main memory 104. Preferably, texture map 105 is a 3D structure, implemented in Cartesian coordinates. Alternatively, map 105 can be a 2D structure, with any required 3D operations implemented reading and writing from and to, respectively, other memory, such as main memory 104.

Detailed Description Text (42):
Frame buffer 99 is a memory structure typically maintained by graphics accelerator 114. Frame buffer 99 preferably

is implemented in relatively fast frame buffer memory 107. Buffer memory 107 can be dedicated memory (as shown in FIG. 1B) or could be one "z" plane of texture memory 109, with frame buffer 99 actually one "z" plane of texture map 105. Alternatively, frame buffer 99 can reside in relatively slower main memory 104, with slower performance as a consequence.

CLAIMS:

19. A method for voxelization of a generalized CSG model of an object in 3D space using a computer system, the computer system including a frame buffer and texture memory, the 3D space including a plurality of coordinate points, the object composed of a plurality of primitives, the generalized CSG model including a generalized CSG tree having a lowest level consisting of a child node associated with each primitive of the object, at least one binary node associated with two child nodes at a level below, with each binary node including an associated Boolean operation that governs the interaction between its associated child nodes, each child node associated with only a single binary node above, a leaf node formed by the binary node at the highest level of the CSG tree and composed of two child nodes below, and each binary node other than the leaf node forming a child node for a binary node above, comprising the steps of: converting the generalized CSG model into a binary 3D model of the object, including the steps of: for each coordinate point in 3D space, creating a classification index in the frame buffer for the point indicative of whether the point is interior to or exterior to each primitive; for each primitive, creating a point classification map in the frame buffer that maps each classification index in the frame buffer against the primitive and its associated child node of the CSG tree; for each child node, determining a point classification map in the frame buffer for its associated binary node based on the child node's point classification map, the point classification map of the other child node associated with its binary node, and the Boolean operation associated with the binary node; for each binary node, determining a point classification map in the frame buffer for its associated binary node above based on the binary node's point classification map, the point classification map of the other node associated with the binary node above, and the Boolean operation associated with the binary node above; and recursively determining the point classification maps for binary nodes until the point classification map in the frame buffer is determined for the leaf node; and voxelizing the binary 3D model of the object; and moving the point classification map for the leaf node into the texture memory.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

---

☐ 20. Document ID: US 6529191 B1

L5: Entry 20 of 34 File: USPT Mar 4, 2003

DOCUMENT-IDENTIFIER: US 6529191 B1
TITLE: Data processing apparatus and data processing method

Brief Summary Text (9):
When the image control data Dgc is delivered from the CPU 1 to the graphic processing system A via the PCI bus 2, it is delivered through a PCI bus interface 21 to be once stored in a 3D graphic temporary buffer 22. Then, the texture data Dt is read out from the 3D graphic temporary buffer 22 and stored in a texture memory 23. The texture data Dt is read out from the texture memory 23 and delivered to a 3D graphic engine 24, according to necessity. The 3D graphic engine 24 performs mapping processing of drawing the inside of a polygon, based on the polygon data Dp and the texture data Dt to produce image data Dg. The produced image data Dg is stored in a frame memory 25. Then, the image data Dg is read out from the frame memory 25 and converted to an analog signal by a RAMDAC 26 to be delivered to a display device, not shown.

Detailed Description Text (14):
Required texture data Dt is read out from the texture and sound-shared memory 6 according to a kind of texture data Dt and address information designated by the polygon data Dp, and then the readout texture data Dt is subjected to mapping according to coordinates of vertices of a polygon to be depicted, to generate image data Dg expanded over a display screen of the display device. The mapping is carried out while the texture data Dt is

subjected to interpolation processing or thinning processing depending upon the inclination of the polygon and required expansion or contraction of the same. In this connection, to represent a transparent object such as a scene through a window, it is required to carry out processing of laying two kinds of pictures, i.e. the window pane and the scene, one over the other. The transparency information included in the polygon data Dp is used in such a case. More specifically, a plurality of the image data Dg corresponding to a certain region on the display screen are generated, the plural image data Dg are each multiplied by a coefficient corresponding to the transparency information, and the plural image data Dg multiplied by the coefficient are synthesized by adding them together into synthesized image data (.alpha. blending processing).

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

---

☐ 21. Document ID: US 6459438 B1

L5: Entry 21 of 34 File: USPT Oct 1, 2002

DOCUMENT-IDENTIFIER: US 6459438 B1
TITLE: Method and apparatus for determining clipping distance

Brief Summary Text (4):
FIG. 1 illustrates a schematic block diagram of a computing system 10 that includes a central processing unit 12, a chipset 14, system memory 16, a video graphics circuit 18, and a display 20. The video graphics circuit 18 includes a raster engine 22 and a frame buffer 24. The system memory 16, for processing video graphics data, includes geometry software 26 and texture maps 28. The display 20 may be a CRT display, LCD display, or other type of computer display.

Detailed Description Text (3):
The present invention can be more fully described with reference to FIGS. 3 to 12. FIG. 3 illustrates a schematic block diagram of a computing system 50 that includes a central processing unit 12, a chipset 14, system memory 16, and a video graphics circuit 52. The system memory 16 includes a memory location for storing texture maps 28. The central processing unit 12 generates a display list 64 that is provided to video graphics circuit 52. The display list 64 maybe generated in accordance with the Open GL specification. In essence, the display list 64 includes a list of commands that provide instructions for the video graphics circuit 52 to render a drawing. Alternatively, or in addition to providing the display list 64, the central processing unit 12 may utilize an immediate mode graphics process wherein each time the central processing unit 12 issues a draw command, the video graphics circuit 52 renders the corresponding drawing. Such an immediate mode graphics process is in accordance with the Open GL specification.

CLAIMS:

10. The apparatus of claim 8, wherein V1, V2, and V3 represents the original vertices, wherein d1, d2, and d3 represent the clipping distance for V1, V2, and V3, respectively, wherein C0 represents the vertex, wherein a.sub.C0, b.sub.C0, and c.sub.C0 represent the barycentric coordinates of the vertex, wherein the memory further comprises operational instructions that cause the processing module to determine the clipping distance to equal d1*a.sub.C0 +d2*b.sub.C0 +d3*c.sub.C0.

12. The apparatus of claim 11, wherein C1 represents the second vertex, wherein a.sub.C1, b.sub.C1, and c.sub.C1, represent the barycentric coordinates of the second vertex, wherein the memory further comprises operational instructions that cause the processing module to determining the second clipping distance to equal d1*a.sub.C1 +d2*b.sub.C1 +d3*c.sub.C1.

14. The apparatus of claim 13, wherein V1 and V2 are within the clipping plane and V3 is outside the clipping plane, wherein d1.sub.2, d2.sub.2, and d3.sub.2 represent the clipping distance for V1, V2, and V3 with respect to the second clipping plane, respectively, wherein C2 represents the third vertex, wherein a.sub.C2, b.sub.C2, and c.sub.C2 represent the barycentric coordinates of the third vertex, wherein the memory further comprises operational instructions that cause the processing module to determine the third clipping distance to equal d1.sub.2

*a.sub.C2 +d2.sub.2 *b.sub.C2 +d3.sub.2 *c.sub.C2.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

22. Document ID: US 6426747 B1

L5: Entry 22 of 34 File: USPT Jul 30, 2002

DOCUMENT-IDENTIFIER: US 6426747 B1
**** See image for Certificate of Correction ****
TITLE: Optimization of mesh locality for transparent vertex caching

Brief Summary Text (5):
In the traditional polygon-rendering pipeline, the graphics processor accesses two types of information from memory: (1) a model geometry and (2) raster images (e.g., texture map, bump map, environment map) used in shading this geometry. The problem of reducing texture image bandwidth is described in Hakura and Gupta, The Design And Analysis Of A Cache Architecture For Texture Mapping, Proceedings of the 24th International Symposium on Computer Architecture (June 1997), 108-120.

Detailed Description Text (8):
A block diagram of an exemplary system architecture to which the invention is directed is shown in FIG. 3. A CPU 2 is connected by a bus 5 to a system/video memory 7. The system / video memory 7 is connected by bus 13 to a graphics processor 14. The graphics processor 14 includes a vertex cache 15, a texture cache 16, and a frame buffer 17. The frame buffer 17 is coupled to a display, such as a CRT 18. The present invention is adapted for use in an interactive computer graphics system in which geometric objects are represented by a mesh structure (as shown, for example, in FIG. 1). The mesh structure is made up of vertices that define triangular faces of the mesh. In such a system, the connectivity of the vertices is fixed whereas the geometry of the mesh structure (the 3D coordinates of the vertices) is not fixed. The vertex coordinates are stored in system (or video) memory 7 in a vertex buffer 8 along with index data 9 defining the order in which the vertices are rendered by a graphics processor. The vertex buffer 8 is a data array of the vertices (i.e., the 3D coordinates of the respective vertices). This arrangement is designed to enhance the speed at which objects can be rendered, because speed is of prime importance in an interactive system. To further enhance rendering speed, the graphics processor 14 includes a cache 15 for storing a subset of the vertex data during rendering.

Detailed Description Text (9):
The bus 13 between the system/video memory 7 and the graphics processor 14 in FIG. 3 is not fast. The present invention minimizes the amount of traffic that has to go over the bus 13. The vertex cache 15 holds a certain number of vertices (e.g., 16 entries of the vertex buffer 8). The texture cache 16 stores images that are textured or mapped onto the mesh geometry.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

23. Document ID: US 6304952 B1

L5: Entry 23 of 34 File: USPT Oct 16, 2001

DOCUMENT-IDENTIFIER: US 6304952 B1
TITLE: Information processing apparatus and information processing method

Detailed Description Text (17):
The polygon definition information comprises drawing area setting information and polygon information. The drawing area setting information includes offset coordinates in the frame memory 58 of a drawing area, that is, a frame memory address of the drawing area, and coordinates of a drawing clipping area for canceling an operation to draw a drawing range indicated by a polygon with coordinates thereof existing outside the drawing area. On the other hand, the polygon information includes polygon attribute information and vertex information. Here, the polygon attribute information is information used for specifying a shading mode, an ALPHA blending mode and a texture mapping mode. On the other hand, the vertex information is information on coordinates in a vertex drawing area, coordinates in a vertex texture area and the color of a vertex, to mention a few.

CLAIMS:

8. The entertainment apparatus as set forth in claim 7 wherein said drawing area setting information includes offset coordinates in a frame memory of a drawing area, and coordinates of a drawing clipping area for canceling an operation to draw a drawing range indicated by a polygon with coordinates thereof existing outside said drawing area.

15. The entertainment apparatus as set forth in claim 1 wherein said graphic processing unit has a frame memory for use as a texture memory, and executes texture mapping processing to stick as a texture a pixel image in said frame memory onto a polygon.

41. The entertainment apparatus as set forth in claim 40 wherein said drawing area setting information includes offset coordinates in a frame memory of a drawing area, and coordinates of a drawing clipping area for canceling an operation to draw a drawing range indicated by a polygon with coordinates thereof existing outside said drawing area.

48. The entertainment apparatus as set forth in claim 29 wherein said graphic processing unit has a frame memory for use as a texture memory, and executes texture mapping processing to stick as a texture a pixel image in said frame memory onto a polygon.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

---

☐ 24. Document ID: US 6204863 B1

L5: Entry 24 of 34 File: USPT Mar 20, 2001

DOCUMENT-IDENTIFIER: US 6204863 B1
TITLE: Method for dynamic XY tiled texture caching

Brief Summary Text (9):
Most graphics subsystems store texture maps in main system memory. Storing the maps in main memory may require a graphics drawing engine to access the texture maps via a bus external to the graphics processor. Each such access and transfer of the texture maps results in processing delays due to inherent memory latency. These latencies substantially slow down the rate at which the graphics processor can therefore process the texture maps.

Detailed Description Text (21):
TCC 325 is also capable of generating a UV tile fetch request to a XY memory control interface. TCC 325 preferably controls UV address requests to and from SRAM 238 by the texture engine 220. Upon receiving the UV address requests, TCC 325 further caches the texture map corresponding to each coordinate for each polygon rendered. TCC 325 always fetches complete UV tiles of texture when a cache-miss occurs instead of just a single texel. Caching UV tiles allows the graphics processor 150 to take advantage of the high burst rate fills to load the cache ways in SRAM 238. SRAM 238 is coupled to TCC 325 to store UV tile chunks of the texture memory.

CLAIMS:

1. A method for dynamically caching a texture map as a tile of texels into an internal memory system of a graphics processor, comprising the steps of:

designating bit values of a new UV texel address to represent a number of texels for a tile and other bit values of the new UV texel address to represent an UV tile address of the tile;

receiving, by a graphics processor having at least a polygon engine, a texture engine with at least a texture cache controller and tag registers, and the internal memory system, a request for the new UV texel address of a new texture map;

determining the UV tile address from the new UV texel address by truncating, by the texture cache controller, the bit values that represent the number of texels for the tile from the new UV texel address so that the other bit values that represent the UV tile address remain;

comparing, by the texture engine, the UV tile address with UV tile addresses latched into the tag registers from previous UV texel address requests; and

in response to the UV tile address not being among latched UV tile addresses of previous UV texel address requests, latching, by the texture engine, the UV tile address into one of the tag registers and retrieving and caching, by the texture engine, the tile at the UV tile address from an external memory system into the internal memory system.

7. A graphics system for rendering texture map information representative of graphics primitives on a computer display, comprising:

a host processor for generating display list information of parameter values defining primitives;

an external system memory coupled to said host processor for storing the display list information; and

a graphics processor coupled to the host processor and the system memory for processing the texture map information;

wherein the graphics processor has at least a polygon engine, a texture engine with at least a texture cache controller and tag registers, and an internal memory system;

wherein bit values of a new UV texel address are designated to represent a number of texels for a tile and other bit values of the new UV texel address are designated to represent an UV tile address of the tile;

wherein the graphics processor receives a request for the new UV texel address of a new texture map;

wherein the UV tile address is determined from the new UV texel address by the texture cache controller truncating the bit values that represent the number of texels for the tile from the new UV texel address so that the other bit values that represent the UV tile address remain;

wherein the texture engine compares the UV tile address with UV tile addresses latched into the tag registers from previous UV texel address requests; and

wherein, in response to the UV tile address not being among latched UV tile addresses of previous UV texel address requests, the texture engine latches the UV tile address into one of the tag registers and the texture engine retrieves and caches the tile at the UV tile address from the external memory system into the internal memory system.

☐ 25. Document ID: US 6151035 A

L5: Entry 25 of 34 File: USPT Nov 21, 2000

DOCUMENT-IDENTIFIER: US 6151035 A
TITLE: Method and system for generating graphic data

Brief Summary Text (13):
The graphic data display system described in claim 7 is provided with a graphic data generation circuit for generating the first graphic data by performing texture mapping processing based on polygon data, a graphic data generation circuit for generating the second graphic data by performing reduction processing on the first graphic data, a memory for storing the second graphic data, and a display device for displaying the second graphic data stored in the memory.

Detailed Description Text (18):
A polygon definition information comprises a drawing region setting information and polygon information. The drawing region setting information comprises an offset coordinate in the frame buffer address of the drawing region and an drawing clipping region coordinate. The drawing clipping region coordinate is served to cancel drawing when the coordinate of a polygon is located outside the drawing region. A polygon information comprises a polygon attribute information and vertex information. The polygon attribute information is an information for specifying shading mode, a blending mode, and texture mapping mode. The vertex information is an information for specifying coordinate in the vertex drawing region, coordinate in vertex texture region, and vertex color.

Detailed Description Text (29):
The frame memory 58 is used as a frame memory and texture memory, and stores the polygon drawing data and texture data. The texture data stored in the frame memory 58 are read as desired by the texture mapping section 93, and the read texture data is used for placing it on a polygon.

Detailed Description Text (31):
An example in which a triangle is drawn is described with reference to FIG. 7. FIG. 7 shows the drawing region 58-1 (refer to FIG. 12) of the frame memory 58, coordinates of the upper left corner, upper right corner, lower left corner, and lower right corner are respectively (0,0), (1280,0), (0,960), and (1280,960), wherein the unit is a pixel. As shown in FIG. 7, in the case that a triangle having vertexes A0 (XA0, YA0), B0 (XB0, YB0), and C0 (XC0, YC0) is drawn, first the respective coordinates of vertexes A0, B0, and C0 are given to the PP 91 as input data. The vertex which is located in the left (in this example, vertex A0) out of top vertexes on the screen is used as the starting vertex. The vertex which is located in the left out of vertexes excepting the starting vertex (in this example, vertex B0) is assigned as the second vertex, and the residual vertex (in this example, C0) is assigned as the third vertex. When, the PP 91 calculates the gradient "a" of the line drawn from the starting vertex to the second vertex (displacement of x-coordinate when y-coordinate is displaced by 1), a gradient "b" of the line drawn from the starting vertex to the third vertex, and a gradient "c" of the line drawn from the second vertex to third vertex, and these gradient values "a", "b", and "c" are set respectively to the DDA 92. For the purpose of simplification herein, it is assumed that y-coordinate of the second vertex is the same as y-coordinate of the third vertex.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KMC | Draw Desc | Image

☐ 26. Document ID: US 5877779 A

L5: Entry 26 of 34 File: USPT Mar 2, 1999

DOCUMENT-IDENTIFIER: US 5877779 A
TITLE: Method and apparatus for efficient rendering of three-dimensional scenes

CLAIMS:

10. A method for performing 3D texture mapping of a given representation of a graphical object, wherein said given representation of said graphical object includes a first polygon, wherein said texture mapping is performed in a computer system including a host processor coupled to a memory controller by a system bus, wherein said computer system further includes a system memory coupled to said memory controller, said method comprising:

storing 3D geometry data and 3D texture data corresponding to said graphical object into a dedicated portion of system memory, wherein said storing is performed by a graphics controller sub-unit included within said memory controller, wherein said dedicated portion of system memory is accessible only by said graphics controller sub-unit, and wherein said 3D texture data includes volume elements corresponding to a texture cube in 3D texture space which includes said graphical object;

generating span data for a first span of said first polygon, wherein said generating is performed by said host processor;

transferring said span data for said first span of said first polygon from said host processor to said graphics controller sub-unit;

mapping said span data for said first span of said first polygon to corresponding volume elements included in said 3D texture data which is stored in said dedicated portion of system memory, wherein said mapping said span data is performed by said graphics controller sub-unit;

rendering said first span of said first polygon according to said corresponding volume elements within said 3D texture data, wherein said rendering is performed by said graphics controller sub-unit.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Drawi Desc | Image

---

☐ 27. Document ID: US 5859646 A

L5: Entry 27 of 34 File: USPT Jan 12, 1999

DOCUMENT-IDENTIFIER: US 5859646 A
TITLE: Graphic drawing processing device and graphic drawing processing system using thereof

Brief Summary Text (13):
As shown in FIG. 3, starting from the initial drawing point on the two dimensional coordinates, the input of data from the frame memory 5 is displayed while the pixels are increased one by one in the x direction, and a one pixel offset is increased in the y-direction if being outside of the triangle.

Detailed Description Text (39):
In the third embodiment of the graphic drawing processing device having the configuration feature of the cache memories 1137 and 1138, comparator 1131, the multiplexer 1133, and the shifters 1135 and 1136, the drawing operation when drawing a polygon (for example, a triangle) while performing texture mapping is as follows. In the same manner as in the second embodiment, the coordinates for the drawing space wherein the pixels are drawn are converted to coordinates for the source space for the texture patterns, using the coordinate conversion section 919. The tag data is then compared with the outputs of the cache memory 1137 and the cache memory 1138 respectively using the comparator 931. Whichever of the cache memory outputs is in agreement with the tag data then becomes the output. When there is a cache miss (the read operation is a miss) in either of these cache memories, the texture pattern data is read out of the frame memory 5 and stored in either one of the cache memories 1137 and 1138. After all the texture patterns to be used are stored in the cache memory 921, the operation section 915 for carrying out pixel unit operations multiplies the frame data obtained from the first and second digital differential analysis section 913 by the texture pattern, using these texture patterns, during the

drawing of the polygon (during the write-in to the drawing data area of the frame memory 5), and the result is written into the drawing data area of the frame memory 5. In the third embodiment, it is possible to perform high speed texture mapping in the same manner as in the first and second embodiments, as shown in FIGS. 5, 6, and 9, and, at the same time, it is possible to increase a degree of freedom of the texture mapping operation by the configuration of the cache memories 1137 and 1138.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

---

28. Document ID: US 5844576 A

L5: Entry 28 of 34 File: USPT Dec 1, 1998

DOCUMENT-IDENTIFIER: US 5844576 A
TITLE: Tiled linear host texture storage

Abstract Text (1):
A process and implementing computer system for graphics applications in which polygon information, including transparency, color and other polygon characteristics, is organized, stored and transferred in terms of areas or tiled blocks of information in a matrix configuration. The polygon bytes of texel information are organized in an exemplary 8.times.8 matrix row and column format in the graphics subsystem for improved cache-hit efficiency and translated to and from the linear addressing scheme of a host storage device when the host storage is accessed to refill the graphics cache. The bytes comprising the memory tiles of polygon information are arranged such that a complete tile of information is transferred in one burst-mode host memory access to minimize normal multi-line access arbitration and other typical access delays.

Brief Summary Text (8):
Texture maps in memory may be stored in a local, relatively fast, RDRAM memory in a graphics device, or such maps may be stored in the system or host memory. When stored as part of a larger host memory, delays in accessing data from the host texture maps are encountered because of the speed of the memory type itself and also because of the nature of the configuration and access process of the host memory.

Brief Summary Text (9):
With specific reference to computer graphics applications, image texture information, such as color transparency of displayed images, is stored in texture maps. A texture map is a two dimensional array of "texels" consisting of "U" texels in the horizontal direction, and "V" lines of texels in the vertical direction. As a polygon is rendered, texels are fetched from a "texture map", processed for lighting and blending, and then such texels become "pixels" of the polygon. As an image is produced on a display screen, each line of data stored in a frame buffer is sequentially accessed and transferred to the display device to fill-in corresponding sequential lines of pixels on the display. The frame buffer is updated by a draw engine portion of the graphics system, which is, in turn, updated by a texture engine portion of the graphics system. The texture engine accesses a texture map which is usually stored in system or host memory. Each such access and transfer of texture information from host memory has a delay time associated therewith because of the inherent dependence of the storing and accessing process. For each access to a texture map stored in host memory, for example, there is a processing latency, as well as delays due to bus access and host memory access arbitration. Moreover, in graphics systems, the next texel of information needed is frequently not the next linear line of information from memory but rather the next texel in a different direction from the last texel transferred. Because of the nature of the linear storage of information in host memory, whenever a texel of information is required by the graphics engine which is not the next linearly displaced texel of information as stored in the host memory, then a new access to the host memory is required to locate the requested texel.

Detailed Description Text (6):
FIG. 5 shows a block diagram of the basic functional units in an exemplary implementation of the disclosed methodology. In a graphics application, as points are drawn on a screen and polygons are assembled for display, a texture engine generates requests for designated addresses. For example, a Texel Address Generator 501 will generate a texel address request which is sent to a Cache Tile Hit Detection Logic Circuit 503 which is included within a Texture Cache Controller 505. The Detection Logic determines whether or not the requested texel is

already stored in the local graphics Texture Cache 507. The Texture Cache 507 in the present example is a 1K byte cache comprised of sixteen "ways" with each way including 64 bytes. Thus, since each tile of memory includes 64 bytes, the Cache 507 can hold 16 tiles of memory. If the requested texel address is in the Texture Cache 507, then a Texel Out 509 is provided to the requesting graphics circuitry for further processing.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

---

29. Document ID: US 5831640 A

L5: Entry 29 of 34 File: USPT Nov 3, 1998

DOCUMENT-IDENTIFIER: US 5831640 A
TITLE: Enhanced texture map data fetching circuit and method

Abstract Text (1):
A circuit and method for increasing the processing efficiency of texture map data requests within a 3D subunit of a computer controlled graphics display system. The 3D graphics display subsystem includes a polygon engine, a texture map engine and a pixel pipeline. The texture map engine contains a texture map data access (TDA) circuit having a cache controller with a computer readable cache memory for containing recently used texture maps stored in (u,v) coordinate space. The cache controller is limited in handling only n cache miss operations simultaneously. In one embodiment, n is 1. The TDA circuit also contains a texture map address (TMA) FIFO memory unit for storing texture map addresses associated with texture data requests that hit or missed in the cache memory unit. Since the cache controller handles up to n misses, the texture engine stalls when the (n+1).sup.th unprocessed texture request miss is encountered. Therefore, the TMA FIFO at any time contains at most n miss addresses therein. Processing efficiency is increased when a miss is encountered but the TMA FIFO contains unprocessed hit addresses. At this time, simultaneously with the cache controller fetching the texture data for the missed address, it can also advantageously: (1) supply data from the cache memory for the previously encountered and stored hit addresses; and (2) accept new hit addresses into the TMA FIFO thereby effectively avoiding a texture engine stall. This is quite unlike the prior art systems which process no hit addresses upon a texture miss but rather stall the texture engine.

Brief Summary Text (7):
However, the process of texture mapping requires a great demand on the memory capacity of the graphics display system because a lot of texture maps are accessed from memory during a typical display screen update cycle. Since the frequency of the screen update cycles is rapid, the individual polygons of the screen (and related texture map data per polygon) need to be accessed and updated at an extremely rapid frequency requiring great data throughput capacities. In view of the above memory demands, high performance graphics hardware units typically contain low access time cache memory units and cache memory controller units for storing and retrieving texture mapped data at high speeds. With texture caches, as a texture-mapped polygon is processed through the graphics unit, an address check is made by the graphics controller as to whether or not the texture map for the polygon is stored in the texture cache. If the requested memory addresses are not present in the texture cache, the cache controller unit of the prior art system stalls while the desired texture data is obtained from external memory. Usually, there is a long latency (stall) from the cache controller unit sending out the external memory request until the texture data is actually fetched from the external memory.

Detailed Description Text (2):
A circuit and method are described for increasing the processing efficiency of texture map data requests within a computer controlled graphics display system. A 3D graphics display subunit is included in the graphics display system and this subunit includes a polygon engine, a texture map engine, and pixel pipeline. The texture map engine contains a texture map data access (TDA) circuit having a cache controller with a computer readable cache memory for containing recently used texture maps stored in (u,v) coordinate space. The cache controller is limited in handling only n cache miss operations simultaneously. In one embodiment, n is 1. The TDA circuit also contains a texture map address FIFO memory unit for storing texture map addresses associated with texture data requests that hit or missed in the cache memory unit. Since the cache controller handles up to n misses, the texture engine stalls when the (n+1).sup.th unprocessed texture request miss is encountered. Therefore, the TMA FIFO at any time

contains at most n miss addresses therein. Processing efficiency is increased when a miss is encountered but the TMA FIFO contains unprocessed hit addresses. At this time, simultaneously with the cache controller fetching the texture data for the missed address, it can also advantageously: (1) supply data from the cache memory for the previously encountered and stored hit addresses; and (2) accept new hit addresses into the TMA FIFO thereby effectively avoiding a texture engine stall. This is quite unlike the prior art systems which process no hit addresses upon a texture miss but rather stall the texture engine.

Detailed Description Text (12):
The host computer system 112 provides data and control signals via bus 100 to a graphics hardware unit or system, e.g., "graphics card" 108. The graphics hardware system 108 contains a 3D graphics subunit 109 for executing a series of display instructions found within a display list stored in computer memory. The display list generally contains instructions regarding the rendering of several graphic primitives, e.g., individual points, lines, polygons, fills, BIT BLTs (bit block transfers), textures, etc. Many of the polygon display instructions include texture data to be displayed within the polygon. Texture data is stored in computer readable (e.g., volatile) memory units of system 112, or local frame buffer 110) in the form of raster based data (e.g., in one form its bit mapped) stored in (u,v) coordinates. The individual components (e.g., "texels") of the texture data are read out of memory and applied within their polygon in particular fashions depending on the placement and perspective of their associated polygon. The process of rendering a polygon with associated texture data is called "texture mapping." Texture mapping requires a great demand on the memory capacity of the computer system 112 because many texture maps are accessed from memory to construct a displayed frame. Since screen updates need to be performed rapidly, polygons need to be updated very rapidly and further texture maps need to be accessed and applied in extremely rapid fashion, increasing memory demands. The graphics hardware system 108, over bus 100", supplies data and control signals to a local frame buffer memory 110 which refreshes the display device 105 for rendering images (including graphics images) on display device 105. Components of the graphics hardware system 108 (e.g., the 3D graphics subunit 109) are discussed in more detail below.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

---

30. Document ID: US 5828382 A

L5: Entry 30 of 34 File: USPT Oct 27, 1998

DOCUMENT-IDENTIFIER: US 5828382 A
TITLE: Apparatus for dynamic XY tiled texture caching

Abstract Text (1):
A graphics subsystem includes hardware for permitting tile texture data to be dynamically cached internally within the hardware. In addition, the system generates a SHIFT signal to permit automatic adjustment of tile texture parameters to facilitate retrieval of the cached texture maps. The system includes a 1 kbyte static random access memory internally disposed within a graphics processor to facilitate UV caching of the texture maps by the graphics processor. A cache controller also disposed within the graphics processor facilitates tile requests by other resources in the graphics subsystem to the internal static random access memory. The cache controller performs UV tile read hit comparisons and subsequent UV to linear address conversions to read texels from the internal static random access memory.

Brief Summary Text (9):
Most graphics subsystems store texture maps in main system memory. Storing the maps in main memory may require a graphics drawing engine to access the texture maps via a bus external to the graphics processor. Each such access and transfer of the texture maps results in processing delays due to inherent memory latency. These latencies substantially slow down the rate at which the graphics processor can therefore process the texture maps.

Detailed Description Text (21):
TCC 325 is also capable of generating a UV tile fetch request to a XY memory control interface. TCC 325 preferably controls UV address requests to and from SRAM 238 by the texture engine 220. Upon receiving the UV address requests, TCC 325 further caches the texture map corresponding to each coordinate for each polygon

rendered. TCC 325 always fetches complete UV tiles of texture when a cache miss occurs instead of just a single texel. Caching UV tiles allows the graphics processor 150 to take advantage of the high burst rate fills to load the cache ways in SRAM 238. SRAM 238 is coupled to TCC 325 to store UV tile chunks of the texture memory.

CLAIMS:

1. A graphics subsystem for rendering texture information representative of graphics primitive on a computer display, comprising:

a host processor for generating display list information of parameter values defining said primitives;

a system memory coupled to said host processor for storing said display list information;

a graphics processor coupled to said host processor and said system memory for processing said texture map information;

wherein said graphics processor includes:

a register file for storing said display list of parameter values;

a polygon engine coupled to said register file for generating polygons responsive to said primitives; and

a texture control unit coupled to said register file for receiving said texture information and generating texture maps representative of said graphics primitives, comprising:

a texture engine for receiving the initial and incremental values of a texture to be fetched from said system memory, said texture engine further receiving polygon size information from said register file in order to track the exact number of texels to complete a primitive to be rendered; and

a static random access memory device disposed within said texture control unit for storing texture maps used to fill in polygons drawn by said polygon engine.

4. In a computer system having a graphics processor for processing graphics information, said graphics processor comprising:

a register file internally disposed within said graphics processor to receive display list information defining graphics primitives to be rendered in said graphics processor;

a polygon engine coupled to said register file to receive the initial and incremental values required to fully specify said primitive to be rendered;

a texture engine coupled to said polygon engine to receive the initial and incremental values required to specify a texture map; and

an internal memory device disposed within said graphics processor and coupled to said texture engine to internally store blocks of said texture map within said graphics processor in a tiled linear format.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

31. Document ID: US 5790130 A

L5: Entry 31 of 34 File: USPT Aug 4, 1998

DOCUMENT-IDENTIFIER: US 5790130 A

** See image for Certificate of Correction **
TITLE: Texel cache interrupt daemon for virtual memory management of texture maps

Brief Summary Text (15):
Furthermore, some texture mapping systems are pipelined so that various operations are performed simultaneously on different object primitives. However, a series of MIP maps for a texture can be large. Most systems employ a local memory that is capable of storing only one such large series of MIP maps at a time. Thus, when there is a switch in the texture used in rendering primitives, the system must download a new series of MIP maps. Typically, the data path used to load the new texture data into the local memory in the texture mapping hardware includes passing through the system's primitive rendering pipeline. Therefore, when a new texture is to be mapped, the primitive rendering pipeline must be allowed to empty out before the new series of MIP maps can be downloaded. Once the series of MIP maps is downloaded, the pipeline must again be filled. The necessity of flushing the primitive rendering pipeline each time a new texture is required reduces the system's bandwidth.

Brief Summary Text (22):
When a graphics API provides a low level software command to render a texture mapped polygon using a particular texture that also is provided, the graphics hardware driver translates that low level command into hardware commands recognizable by the hardware device 150. The graphics hardware driver then provides the commands to render the polygon to the hardware device 150 by writing to appropriate registers and input buffers within the hardware device. Additionally, the graphics hardware driver provides the texture data for that particular polygon to the local memory 155 of the hardware device 150 where it is temporarily stored during rendering of that polygon. As is described herein, prior art computer graphics systems download the entire series of MIP maps for a single texture from the graphics hardware driver to the local memory of the hardware device each time that texture is required to render an image or component of the image.

Detailed Description Text (3):
The present invention relates to a software daemon that runs on the processor of a computer graphics system host computer and manages the storage of texture data within the local memory of a texture mapping graphics hardware device connected to the host computer. The daemon manages the local memory in such a way that the memory appears virtual. By contrast with prior art systems that download entire series of texture MIP maps into the local memory of the hardware device, the software manager of the present invention downloads only portions of texture MIP maps necessary at any one time to render an image or image portion. The manager considers the recent past frequency of use of particular texture map portions as well as the predicted future use of such map portions (based on relative priorities of the textures) in determining which portions to replace in the local hardware memory when new texture data is required by the hardware device to render an image or portion thereof.

Detailed Description Text (25):
FIG. 4 is a block diagram of one embodiment of a graphics system hardware device with which the software of the invention can be used. The device includes texture mapping hardware having a cache memory for storing texture data locally. It should be understood that the illustrative implementation shown is merely exemplary with respect to the number of boards and chips, the manner in which they are partitioned, the bus widths, and the data transfer rates. Numerous other implementations can be employed. As shown, the system includes a front end board 10, a texture mapping board 12, and a frame buffer board 14. The front end board communicates with a host computer 15 over a 52-bit bus 16. The front end board receives primitives to be rendered from the host computer over bus 16. The primitives are specified by x,y,z vector coordinate data, R,G,B color data and texture S,T coordinates for portions of the primitives, such as for the vertices when the primitive is a triangle. Data representing the primitives in three dimensions then is provided by the front end board 10 to the texture mapping board 12 and the frame buffer board 14 over 85-bit bus 18. The texture mapping board interpolates the primitive data received to compute the screen display pixels that will represent the primitive, and determines corresponding resultant texture data for each primitive pixel. The resultant texture data is provided to the frame buffer board over a five 55-bit buses 28, which is shown in FIG. 4 as a single bus to clarify the figure.

Detailed Description Text (31):
The texture mapping chip 46 successively receives primitive data over bus 18 representing the primitives to be rendered on the display screen. As discussed above, the 22 primitives provided from the 3-D geometry accelerator chips 32A-C include points, lines and triangles. The texture mapping board does not perform texture mapping of points or lines, and operates only upon triangle primitives. The data representing the triangle primitives includes the x,y,z object pixel coordinates for at least one vertex, the object color R,G,B values of the at least one vertex, the coordinates in S,T of the portions of the texture map that correspond to the at least one vertex, and the plane

equation of the triangle. The texture mapping chip 46 ignores the object pixel z coordinate and the object color R,G,B values. The chip 46 interpolates the x,y pixel coordinates and interpolates S and T coordinates that correspond to each x,y screen display pixel that represents the primitive. For each pixel, the texture mapping chip accesses the portion of the texture MIP map that corresponds thereto from the cache memory, and computes resultant texture data for the pixel, which may include a weighted average of multiple texels.

Detailed Description Text (32):
In one exemplary embodiment, the cache stores sixty-four blocks of 256.times.256 texels. Unlike the local memory employed in the texture mapping hardware of prior art systems, the cache memory of the present invention may not store the entire series of MIP maps of the texture that maps to the primitive being rendered, such as for large textures. Additionally, unlike in prior art systems, the cache memory may not store the series of MIP maps for multiple textures that map to the primitive being rendered. Rather, the cache memory stores at any one time only the particular portions of the series of MIP maps actually used in currently rendering the primitive. Therefore, for most applications, only a portion of the complete texture data for the image being rendered will be stored in the cache memory at any one time.

Detailed Description Text (33):
The complete series of MIP maps for each texture is arranged and stored in the main memory 17 of the host computer 15. The TIM of the present invention runs on the processor of the host computer. For each pixel of the primitive being rendered, the texture mapping chip 46 accesses a directory of the cache memory 48 to determine whether the corresponding texel or texels of the texture MIP maps are currently present in the cache. If the corresponding texels are stored in the cache memory at the time of the access, a cache hit occurs, and the texels are read from the cache and operated upon by the texture mapping chip 46 to compute the resultant texture data which is passed to the frame buffer board.

Detailed Description Text (35):
The requested texture data is retrieved by the TIM from its main memory and is downloaded to the texture mapping board 48 over bus 24, bypassing the 3-D primitive rendering pipeline through the front end board and the texture mapping chip. Thus, when a cache miss interrupt occurs, the front end board can continue to operate upon 3-D primitives and provide output primitive data over bus 18 to the texture mapping chip and the frame buffer board, while the texture data associated with a primitive that caused the cache miss is being downloaded from main memory 17. In contrast to conventional texture mapping systems, the downloading of texture data to the texture mapping hardware does not require a flushing of the 3-D primitive pipeline, thereby increasing the bandwidth and performance of the system.

Detailed Description Text (36):
The resultant texture data for each pixel is provided by the texture mapping chip 46 to the frame buffer board over five buses 28. The five buses 28 are respectively coupled to five frame buffer controller chips 50A, 50B, 50C, 50D and 50E provided on the frame buffer board, and provide resultant texture data to the frame buffer controller chips in parallel. The frame buffer controller chips 50A-E are respectively coupled to groups of associated VRAM (video random access memory) chips 51A-E. The frame buffer board further includes four video format chips, 52A, 52B, 52C and 52D, and a RAMDAC (random access memory digital-to-analog converter) 54. The frame buffer controller chips control different, non-overlapping segments of the display screen. Each frame buffer controller chip receives primitive data from the front end board over bus 18, and resultant texture mapping data from the texture mapping board over bus 28. The frame buffer controller chips interpolate the primitive data to compute the screen display pixel coordinates in their respective segments that represent the primitive, and the corresponding object R,G,B color values for each pixel coordinate. For those primitives (i.e., triangles) for which resultant texture data is provided from the texture mapping board, the frame buffer controller chips combine, on a pixel by pixel basis, the object color values and the resultant texture data to generate final R,G,B values for each pixel to be displayed on the display screen.

Detailed Description Text (54):
During rendering, the tiler/boundary checker 72 generates a read cache tag for the block of texture data that maps to the pixel to be rendered. The manner in which the cache block tag and the read cache tag are generated is explained in more detail below. The tags are 23-bit fields that include eight bits representing the texture ID of the texture data, a bit used in determining the map number of the texture data, and the seven high-order S and T coordinates of the texture data. The cache directory 78 compares the read cache tag provided from the tiler/boundary with the block tags stored in the directory to determine whether the block of texture data to be used in rendering is in the cache memory. If the block tag of the texture data that maps to the primitive to be rendered is stored in (i.e., hits) the cache directory, then the cache directory generates a block index that indicates the physical

location of the block of texture data in the cache that corresponds to the hit tag. The computation of the block index is discussed in greater detail below. A texel address is also generated by the tiler/boundary checker 72 for each texel to be read from the cache and indicates the location of the texel within the block. The texel address includes low-order address bits of the interpolated S,T coordinates for larger size maps, and is computed based on an algorithm described below for smaller size maps. The block index and texel address together comprise the cache address which indicates the location of the texel within the cache. As is described in greater detail below, the LSBs of the S and T coordinates are decoded to determine in which of four cache interleaves the texel is stored, and the remaining bits of the cache address are provided to the texel cache access circuit 82 along with a command over line 84 to read the texel data stored at the addressed location in the cache.

Detailed Description Text (55):
If the read cache tag does not match any of the block tags stored in the cache directory 78, a miss occurs and the cache directory 78 generates an interrupt control signal over line 94 (FIG. 4) to the distributor chip 30 on the front end board, which generates an interrupt over line 95 to the host computer 15. In response to the interrupt, TIM, discussed in more detail below, which reads the missed block tag from the cache directory and downloads the corresponding block of texture data into the cache memory in a manner that bypasses the 3-D primitive pipeline in the front end board 10 and the texture mapping chip 46. The texture data downloaded from TIM is provided over bus 24, through the texel port 92 (FIG. 5) to the texel cache access circuit 82, which writes the data to the SDRAMs that form the cache memory.

Detailed Description Text (57):
During rendering, the later stages of the pipeline in the frame buffer board 14 do not proceed with processing a primitive until the texture data corresponding to the primitive is received from the texture mapping board. Therefore, when a cache miss occurs and the texture mapping chip waits for the new texture data to be downloaded, the frame buffer board 14 similarly waits for the resultant texture data to be provided from the texture mapping chip. As with the texture mapping chip, the stages of the pipeline that follow the stage that receives the texture mapping data continue to process those primitives received prior to the miss primitive, and the stages of the pipeline that precede the stage that receives texture mapping data also continue to process primitives unless and until the pipeline fills up.

Detailed Description Text (58):
It should be understood that when the pipeline of either the texture mapping board or the frame buffer board backs up when waiting for new texture data in response to a cache miss, the pipeline in the front end board 10 will similarly back up. Because cache misses will occur and will result in an access to the host computer main memory and a downloading of texture data that will take several cycles to complete, it is desirable to ensure that the pipeline in the texture mapping chip never has to wait because the pipeline in the frame buffer board has become backed up. Therefore, in one embodiment, the frame buffer board is provided with a deeper primitive pipeline than the texture mapping board, so that the texture mapping pipeline should not be delayed by waiting for the frame buffer pipeline to become available.

Detailed Description Text (59):
In one embodiment, the capability is provided to turn off texture mapping. This is accomplished by TIM operating on the processor 19 of the host computer to set a register in both the texture mapping board 12 and the frame buffer board 14. When set to turn off texture mapping, these registers respectively inhibit the texture mapping chip 46 from providing texture data to the frame buffer board 14, and instruct the frame buffer board to proceed with rendering primitives without waiting for texture data from the texture mapping board.

Detailed Description Text (154):
As discussed above, one feature of the hardware device enables a MIP map for a new texture to be downloaded to the local memory in the texture mapping hardware through a data path that is separate from the pipeline for handling 3-D primitive data. Referring to the illustrative embodiment disclosed in the figures, the texture mapping board 12 (FIG. 4) and the texture mapping chip 46 (FIG. 5) each has separate ports for respectively receiving 3-D primitive data and texture data. The 3-D primitive data is received from the concentrator chip 36 via bus 18, whereas the texture data is received from the 2-D geometry accelerator chip 34 via bus 24. Therefore, when new texture data is downloaded from the host computer 15 by TIM to the texture mapping chip 46, the 3-D primitive pipeline through the front end board 10 and the texture mapping chip 46 need not be flushed, thereby providing increased bandwidth when compared with conventional texture mapping systems which require a flushing of the 3-D primitive pipeline whenever new texture data is downloaded to local memory in the texture mapping hardware.

Detailed Description Text (156):

As discussed above, in one embodiment shown in FIG. 4A, portions of the graphics system are duplicated to increase system bandwidth. The texture mapping board 12 is provided with two texture mapping chips 46A and 46B, and two cache memories 48A and 48B. In this embodiment, both cache memories 48 maintain the same texture data at all times, because both the two texture mapping chips typically operate simultaneously on primitives using the same texture data. Therefore, by updating both caches anytime a miss is received from one, this embodiment conserves system bandwidth by ensuring that the same texture data need not be downloaded in separate operations to the two caches.

Detailed Description Text (170):
In one embodiment, a capability is provided to disable the cache operation of the local memory 48 on the texture mapping board by disabling cache misses, so that texture data for any 3-D primitive is downloaded into the memory 48 before it is required during rendering of the primitive. Each texture mapping chip 46 includes a status bit indicating that operation of its local memory as a cache is enabled. When this status bit is asserted, cache misses result in an interrupt of the host computer, and a halting of the texture mapping chip. However, when the status bit is deasserted, the local memory 48 on the texture mapping board does not operate as a cache, and the texture data for any primitive is downloaded into the memory 48 before it is needed by the primitive so that misses to the memory do not occur. In one embodiment of the invention, when the operation of the local memory as a cache is disabled, texture data is download ed by TIM to the local memory on the texture mapping board through the 3-D primitive pipeline to facilitate synchronization of the texture data with the corresponding 3-D primitive data.

Detailed Description Text (235):
FIG. 26 is a flow chart illustrating the operation of the TIM software daemon of the present invention. Shown in the upper portion of the flow diagram are the steps of the device independent operation of TIM and shown in the lower portion of the flow diagram are the steps of the device dependent operation of TIM. Also shown, connected to the device dependent routines of TIM, is a graphics hardware device 660 that renders texture mapped images requested by a user of high level processes (described above). The graphics hardware device 660 could be any number of graphics hardware devices. For simplicity of illustration, it is assumed that the graphics hardware device includes a local cache memory that stores blocks of texture data and for which interrupts are processed. The hardware device may be identical or similar to that described above.

CLAIMS:

22. A software daemon that manages texture data in a texture mapping computer graphics system, the system including a host computer having a system memory that stores texture data, a graphics hardware device, coupled to the host computer, that renders texture mapped images, and including a local memory that stores in blocks at least a portion of the texture data stored in the system memory at any one time, the blocks including portions of texture maps of larger-sized textures, the software daemon comprising:

a process that allocates the texture data among blocks for downloading to the hardware device;

a routine that determines which block is required by the hardware device during an interrupt;

a routine that selects the block within the hardware device to be replaced during the interrupt; and

a routine that downloads the required block to the hardware device to replace the selected block within the hardware device during the interrupt.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

32. Document ID: US 5574836 A

L5: Entry 32 of 34 File: USPT Nov 12, 1996

DOCUMENT-IDENTIFIER: US 5574836 A

TITLE: Interactive display apparatus and method with viewer position compensation

Detailed Description Text (25):
Summarizing, FIG. 3 is a flow chart showing the steps of the process 100 of writing data representing a graphic object into a frame buffer for subsequent display. At step 102, the transform matrix for the current display is determined. At step 104, the coordinate-based graphic primitives of that object are transformed using the transform matrix. At step 106, the transformed graphic primitives are sent to the graphics engine with the display coordinates. Finally, at step 108, the graphics engine uses the display coordinates as a frame buffer address to render the image pixel data of the graphic object into the frame buffer.

Detailed Description Text (58):
Each tile in the collection of tiles allocated to a window will contain a portion of that window. The generally non-contiguous collection of tiles are mapped by the underlay map (see below) to appear to the graphics engine as a contiguous 2 dimensional region of memory.

Detailed Description Text (76):
Underlay map 128 serves three functions in this embodiment. The first is to allow use of off-the-shelf graphics engines and memory. The addition of underlay map 128 provides an enhancement to the addressing mechanism of graphics rendering engine 120 allowing it to render to a buffer substantially larger than 1 Mbyte. The second function provides a means to allocate and deallocate tiles for use by graphics engine 60. The third function provides a simple tile mapping mechanism for efficiently accessing non-contiguous tiles from a two-dimensional memory array that are mapped onto a contiguously addressed plane in a three dimensional space.

☐ 33. Document ID: US 5469535 A

L5: Entry 33 of 34 File: USPT Nov 21, 1995

DOCUMENT-IDENTIFIER: US 5469535 A
TITLE: Three-dimensional, texture mapping display system

Abstract Text (1):
A three-dimensional, texture mapping display system for displaying three-dimensional projected polygons with texture map on a two-dimensional raster display screen includes a host processor and a graphics co-processor. The host processor is responsive to input data from a host memory defining an overall image that is to be displayed on a three-dimensional form on a raster display device for generating command signals, screen coordinates on the display device, and corresponding texture space coordinates on a texture memory. The graphics co-processor is responsive to the command signals, screen coordinates, and corresponding screen space coordinates for determining destination addresses for each pixel on the screen space of the display device and for determining a corresponding source address applying surface texture data to the image for each destination address. The graphics co-processor includes a look-up table for storing Z-mapping function values so as to provide delta and acceleration values in texture space which can be used to determine the source addresses.

Brief Summary Text (12):
In accordance with these aims and objectives, the present invention is concerned with the provision of a display system for displaying three-dimensional projected polygons with texture maps on a two-dimensional raster display screen which includes a host processor and a graphics processor. The host processor is responsive to input data from a host memory defining the vertices of a polygonal image that is to be displayed in a three-dimensional form on a raster display means for generating command signals, screen coordinates on the raster display means and corresponding texture space coordinates on a texture memory. The graphics processor is responsive to the command signals, screen coordinates and corresponding texture space coordinates for determining destination addresses for each pixel on the screen space of the raster display means and for determining corresponding source addresses supplying surface texture to the image for each destination address.

Detailed Description Text (2):
Referring now in detail to the drawings, there is shown in FIG. 1 an overall block diagram of an improved three-dimensional, texture mapping display system 10 constructed in accordance with the principles of the present invention. The display system is used to display three-dimensional projected polygons with texture maps on a two-dimensional raster display device. The display system 10 is particularly adapted for use in video games to allow real-time animation of video game scenes with textured plane surfaces at a high speed of operation. The display system is comprised of a host computer 12 which is preferably a digital signal processor (DSP), a graphics co-processor 14, a texture memory 16, a video frame buffer 18, a color look-up table 20, and a cathode ray tube (CRT) or other display device 22.

Detailed Description Text (7):
As can be seen from FIG. 2, there is a detailed block diagram of the graphics co-processor 14 of the present invention for processing of the input data in response to command signals from the host computer 12 so as to generate on the bitmap a polygon filled with data from the texture memory in the form which is synchronized with the Z coordinate values associated with each vertex. The graphics processor 14 includes an arithmetic logic unit (ALU) 25, look-up table 26 for supplying values of ZTABLED and ZTABLEDD, a screen address generator 28, a texture map address generator 30, and a data unit 32.

Detailed Description Text (11):
Then, the X, Y screen coordinates are set in the block 118 to the initial starting coordinates Xsmin, Ysmin. It should be understood that these are the starting coordinates for both "Xmin edge" and "Xmax edge." Then, in block 122 the X screen coordinate is increased by 1 repeatedly to plot the data from the texture memory 30 for each X screen coordinate until the "Xmax edge" is reached. In effect, this plots texture data from the left to the right edge of the current scan line of the polygon.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments   KWIC | Drawi Desc | Image

---

34. Document ID: US 5303321 A

L5: Entry 34 of 34 File: USPT Apr 12, 1994

DOCUMENT-IDENTIFIER: US 5303321 A
TITLE: Integrated hardware generator for area fill, conics and vectors in a graphics rendering processor

Abstract Text (1):
An integrated hardware generator for generating digital signals representative of vectors, polygons and conics primitives and area fills therefor. The primitive signals are used in the formation of a final digital output signal read into a bit map memory of a graphics display processor. Its operation is based on applying one or more of a set of internal subfunctions to generate mathematical solutions for rendering each geometric shape as a graphics primitive digital signal. The basic building block of the generator is a digital differential analyzer which is adapted to accumulate fractional (subpixel) components of x/y coordinate data and to signal when the accumulation overflows across pixel boundaries. This occurrence enables an increment or decrement of the x/y coordinates that indicate the pixel address to be loaded (drawn). The digital differential analyzer forms an independent vector generator and comprises a pair of input differential multiplexers, an arithmetic logic unit and a register file. On receipt of a command from a host processor the multiplexers receive parameters that specify the primitive to be drawn and select the appropriate parameters for input to the arithmetic logic unit. The arithmetic logic unit accumulates parameters and stores the results in the register file. The output of the register file is fed back to the multiplexers to provide inputs fr the next operation.

CLAIMS:

1. An integrated hardware vector, conic and area fill primitive generator for generating digital primitive signals representative of vectors including primitive lines including polygons and conics primitives and area fills therefore, said primitive signals being used in a graphics display processor, said processor further including a display memory

for storing system commands and instructions relating to the generation of said primitives, a host processor for generating said system commands, combinational means for acquiring said primitive signals and combining them with a second set of signal signals comprising symbols and a third set of digital signals comprising background and texturing signals to form a fourth digital signal comprising an output signal, said output signal being written in a predetermined address in a bit map memory of a graphics display, said primitive generator comprising:

input means for acquiring data and instructions to draw a primitive in said bit map memory; and

iterative means coupled to the input means for calculating and drawing said primitives relative to a first address in said bit map memory; said iterative means including

means for determining the number of iterations needed to generate a primitive line.

an address counter for storing the x and y coordinates of a pixel address in the bit map memory to which said primitive line is to be drawn;

a digital differential analyzer for accumulating components of x and y coordinate data, said analyzer comparing received x and y coordinate data with last previous x and y coordinate data stored in said analyzer and changing the x and y coordinates in said address counter when said x and y coordinates indicate that a pixel boundary has been crossed by said primitive vector line;

fetch means for acquiring sine and cosine data from said display memory for use in drawing curved primitive lines for conics primitives;

means for regulating the rates of x and y coordinate data input so that a curved primitive line is drawn;

line generator means for generating fill lines in the space bounded by drawn primitive lines; and

signal means for indicating when a drawn primitive line has reached a second address in said bit map memory and for resetting the iterative means to calculate and draw a new primitive line.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments   KWIC | Draw Desc | Image

Generate Collection | Print

Terms	Documents
L4 and l1	34

Display Format: - Change Format

Previous Page   Next Page